Sebastian Mai

Simultaneous Localisation and Optimisation – Towards Swarm Intelligence in Robots

INF FAKULTÄT FÜR INFORMATIK

Simultaneous Localisation and Optimisation – Towards Swarm Intelligence in Robots

Master Thesis

Author
Sebastian Mai

July 30, 2018

Supervisor: Prof. Dr. Sanaz Mostaghim, Chair of Intelligent Systems

Advisor: Dr. Christoph Steup

Sebastian Mai: *Simultaneous Localisation and Optimisation – Towards Swarm Intelligence in Robots*
Otto-von-Guericke-Universität
Magdeburg, 2018.

Abstract

This thesis presents a generic approach that enables the use of Particle Swarm Optimisation in sensor-based environments: The Simultaneous Localisation and Optimisation method. A lot of promising research has been conducted to overcome problems specific to robotic applications in swarm intelligence applications. However, most research is limited to simulation. Often the localisation of the particles is taken for granted in those simulations, as the choice of a localisation method would reduce the generality of the results. The Simultaneous Localisation and Optimisation method that was developed for this thesis addresses the problem of localisation in swarm robotics. A localisation method inspired by the GPS-free Directed Localisation method is used to convert data from distance and motion sensors to information on particle positions. This information is then used by Standard Particle Swarm Optimisation to compute the next robot movement. The Simultaneous Localisation and Optimisation method encapsulates the swarm intelligence algorithm to work within a sensor-actor-based robot. The experiments conducted in this thesis show that the algorithm works. The localisation is most sensitive to errors in the distance measurements. Additionally, the localisation error can enhance the exploration behaviour of the algorithm. However, the exploitation is limited by additive errors in the sensor measurements if the particles move close to each other. All in all, the Simultaneous Optimisation and Localisation algorithm showed good results and is suited for further development and application.

Contents

List of Figures

List of Tables

Table of Acronyms

CSS Closest Solution Selection, one component of the localisation in the SLO algorithm

GDL GPS-free Directed Localization [2], a localisation algorithm

GPS Global Positioning System [15]

knn K Nearest Neighbours, a neighbourhood relationship

PSO Particle Swarm Optimisation, a family of swarm intelligence algorithms

RTS Round Trip Selection, one component of the localisation in the SLO algorithm

SLAM Simultaneous Localization and Mapping [40]

SLO Simultaneous Localisation and Optimisation, the algorithm/method developed for this thesis

SPSO (2011) One specific PSO algorithm [56]

1 Introduction

1.1 Motivation: Swarm Intelligence in Robotic Applications

Swarm intelligence is a field that promises new applications in robotics. Swarm algorithms are decentralised by nature and offer robustness, scalability and flexibility. While the algorithms are mostly used to solve problems in numeric optimisation, the research in biology that inspires them shows the benefits of swarm behaviour in the physical world. Algorithms such as Particle Swarm Optimisation (PSO), Attraction Repulsion Behaviour and Ant Colony Optimisation are useful in different robotic domains. Applications of such behaviours include collective search [8, 35], collision avoidance [12], formation control [13] and traffic management [9]. Systems using swarm intelligence in robot control exist nowadays [41] for example in path-planning [57], learning [43] or collective search [17]. Some platforms developed for swarm robotics also exist, e.g. the Kilobots [46] or the Kinbots [17].

The main difficulty in applying the algorithms known from swarm intelligence in robotic applications stems from the assumptions of swarm intelligence theory. Those assumptions do not fully apply to robotic applications. In swarm intelligence research knowledge about the environment is usually present in all agents involved in the algorithm. A physical robot in a robotic swarm not have the same knowledge as an agent in a swarm intelligence algorithm. [42, 17]

In this thesis the following assumptions motivate the approach: A robot must gather knowledge about its environment through sensors and can act on the environment through actors. A measurement made by any sensor contains an unknown error. The sensors often do not directly report the desired information. For example, the desired information is the distance covered by the robot, but the robot is only able to measure the turn rate of its wheels. This

distance in turn may be another data item needed to compute the position of the robot. In real world applications the sensor information is subject to additional constraints like limited transmission rates and measurement range. A robot that only perceives its environment through sensors has a different perspective on the environment. This difference in perspective is a key reason why swarm intelligence algorithms are difficult to integrate into robotic applications.

In existing implementations the information is often acquired in a centralised manner. This limits the benefits of swarm algorithms that are decentralised by nature. The purpose of this thesis is to create an algorithm that does not bypass the sensory perception of the environment. A state of the art swarm intelligence algorithm is to be embedded in an algorithm that only relies on sensors to perceive the environment.

In swarm intelligence research often a specific pattern of movement update is used. Each particle is modelled as a point in an n-dimensional space. In each time step of the algorithm a movement is computed for each particle. The exact movement is computed based on the other particles' positions (for more details on swarm movement see subsection 2.1.1). The reference for those positions is a global coordinate systems, shared between all particles. Additionally, a particle must be able to move with the computed velocity. When the swarm intelligence is used as a high-level control mechanism the robots need information about their environment. This information includes the position of other particles and the orientation of the coordinate system. Furthermore, a robot must be able to move according to the movement command specified by the algorithm.

1.2 Goals

The goal of this thesis is to create an algorithm that combines a swarm intelligence algorithm with the sensor data transformation necessary to run it. The algorithm must be able to work within a sensor-actor-based robot and be compatible with many other swarm intelligence algorithms. Additionally, it must be shown that the method works and how the algorithm is affected by errors. To achieve those goals four tasks must be completed: The algorithm must be designed, implemented and an experiment must be performed and analysed.

1.3 Outline

In chapter two the state of research is presented. This includes relevant localisation technologies, the necessary sensors and swarm intelligence algorithms. A description of the algorithm itself is given in chapter three. The fourth chapter explains the experiments. The chapter includes the setup of the experiments and a discussion of the most interesting results. The last chapter concludes with a discussion of potential uses of the algorithm and questions for further research.

2 State of the Art

The goal of this thesis is to decouple a swarm intelligence algorithm from its direct perception of the environment. Instead, the knowledge about the environment must be deduced from sensors. This chapter contains an overview on swarm methods and examines their information need.

The localisation method is a core component of the algorithm. It is difficult to replace it without changing sensors. Hence, different technologies and techniques must be considered for localisation. This is especially important, as localisation is always very specific to the application and there is on standard solution for all use-cases. In the last part of the chapter, different sensors and technologies, that can be used to measure the values required for the chosen localisation algorithm, are examined.

2.1 Swarm Movement

A swarm consists of multiple agents that interact locally. Because of this interaction a global behaviour emerges (i.e. a shared movement or a formation). In swarm intelligence research, the interactions and behaviours of the agents are chosen to achieve a desirable behaviour of the swarm. [13]

Multiple types of swarm behaviour exist. Some behaviours, like Particle Swarm Optimisation (PSO) or Ant Colony Optimisation (ACO) are designed to solve an objective (or multiple objectives) in interaction with the environment. Other behaviours, such as attraction repulsion behaviours, steer the distribution of swarm members in space without solving a task in the environment. A good overview on swarm-based search behaviours in robotic applications is given by Liu et al. [27]. The mostly used swarm algorithms are PSO [56, 41] and ACO [9]. Another important branch of swarm intelligence algorithms are attraction repulsion behaviours [13]. Additionally, some algorithms in swarm

intelligence operate within the sensor-actor pattern (i.e. [17]). Some of those algorithms are developed for the Kilobot platform [46].

The Kilobots are robots that use local interaction to generate emergent behaviour. However, their control algorithms are always specifically designed for the Kilobots. The "higher level" algorithms like PSO are hardly used at all [46]. Ideally, methods such as attraction repulsion behaviour or PSO can directly be used in robotic applications. The difficulty in implementing those algorithms in robotic swarms is the different perspective on the environment. How to obtain the information needed by the PSO from sensor data is be examined in more detail in the next sections.

ACO works on a different movement model than attraction repulsion behaviours and PSO. Local interaction in attraction repulsion behaviours and PSO is based on attractive and repulsive forces between the particles. ACO uses (virtual) pheromone traces in the environment to guide robot decisions [9]. This thesis focusses on PSO, as it is one of the mostly used examples for robot control algorithms in swarm intelligence. PSO is closely related to attraction repulsion behaviour, hence the adoption of those behaviours is straightforward. Changing the algorithm to ACO requires more work, but is also possible in theory.

2.1.1 Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) is one of the major swarm intelligence techniques. Many applications for PSO algorithms exist nowadays [41]. The algorithm Standard Particle Swarm Optimisation 2011 (SPSO 2011) by Zambrano-Bigiarini et al. [56] is the basis for the PSO algorithm used in this thesis. Since SPSO is often used as a reference in comparison to other optimisation algorithms, it was chosen as the algorithm to be implemented. As a typical representative of the PSO algorithms, SPSO is explained here. In PSO a swarm of virtual particles searches for the minimum in a multi-dimensional objective function that assigns a fitness value $f(x)$ for each value x in the search space S. The particles can measure the value of the objective function at their current position. The particles then move according to their knowledge about the fitness of the current and the last step. Each PSO algorithm contains two major movement components: The *social* and the *cognitive* component. The social component moves the particle closer to the previous best position of all

particles in its neighbourhood. The cognitive component moves each particle closer to its own previous best position. As the particles move towards the best known positions the swarm will eventually find an optimum of the fitness function in the search space.

$$\vec{p}_i^t = \vec{X}_i^t + c_1 \vec{U}_1^t \otimes (\vec{P}_i^t - \vec{X}_i^t) \tag{2.1}$$

$$\vec{l}_i^t = \vec{X}_i^t + c_2 \vec{U}_2^t \otimes (\vec{L}_i^t - \vec{X}_i^t) \tag{2.2}$$

$$\vec{G}_i^t = \frac{(\vec{X}_i^t + \vec{p}_i^t + \vec{l}_i^t)}{3} \tag{2.3}$$

$$\vec{V}_i^{t+1} = \omega \vec{V}_i^t + \mathcal{H}_i(\vec{G}_i^t, \parallel \vec{G}_i^t - \vec{X}_i^t \parallel) \tag{2.4}$$

$$\vec{X}_i^{t+1} = \vec{X}_i^t + \vec{V}_i^{t+1} \tag{2.5}$$

Equation 2.1 to Equation 2.5 are used in SPSO 2011 [56]. The position $\vec{X}_i^t$ is updated with a velocity vector $\vec{V}_i^t$ at each time step. The velocity itself is computed as a combination of $\vec{p}_i^t$ the cognitive component and $\vec{l}_i^t$ the social component. In Standard PSO the velocity is randomly selected from a hypersphere around the point $\vec{G}_i^t$ that is computed from the social and cognitive component (Equation 2.3). The inertia $\omega \vec{V}_i^t$ is added to the velocity directly before it is used to update the position (Equation 2.4).

A lot of randomness is included in the movement of the particles. In the computation of $\vec{p}_i^t$ and $\vec{l}_i^t$ the relative positions of the local best (L_i^t) and previous best (P_i^t) are not only scaled by the parameters c_1 and c_2, but also by randomly chosen vectors $\vec{U}_{1,2}^t$. In addition, the velocity is based on the random choice of a point in the hypersphere around G_i. The randomness increases the diversity in the swarm positions and the particles move more because of the random processes. This leads to a better exploration behaviour of the algorithm.

The increase in exploration performance is a common theme in many of the PSO variants [10, 4]. Charged PSO [4] is a PSO variant that uses a repulsion force between the particles. One part of the swarm acts normally, while another part of the swarm is repelled by the other particles. The repulsive forces push the particles apart. Thus, they cover a larger area and the chance of the swarm to converge to a local optimum is decreased. Heterogeneous PSO [10] takes a similar approach. By assigning different behaviours to the particles, their search behaviour is diversified and exploration is increased.

Many topics in current research, for example odour source localisation [18], are robotic applications. A lot of research was conducted in PSO with noisy fitness

functions [25, 39, 43]. This is important as the objective is most likely evaluated by sensors and sensor values are never measured without error. Additionally, the fitness function may change over time resulting in a dynamic optimisation problem [39, 44]. In dynamic problems the goal is not simply to find the optimum in the fitness landscape, but to track a changing optimum in the environment. Jorhedi [44] gives an overview on the challenges and methods for solving dynamic problems with PSO.

Not only the sensors pose a challenge to PSO algorithms. Particles are able to move freely across the search space, but robots are not. Pugh and Alcherio [42] identified two limitations to the robot movement: The distance each robot can travel in each time step and the time needed to change a robot's heading. Those problems can be mitigated by increasing the time used for each generation of the PSO [42] or velocity clamping [45]. Still, path planning within PSO is not well researched. Movement is usually not assumed to be costly in comparison to fitness evaluation. In most robotic applications, however, reading a sensor is cheaper than moving the robot in terms of energy and time usage. An additional problem that is not addressed by PSO research is collision avoidance, which can pose further challenges to path planning. [42]

PSO is often used for path planning like in [57], however, the approach is completely different, as the particles are solutions to the decision making of the robot and are not equivalent to robot positions. Nevertheless, in some scenarios PSO is considered as a method with the particle-robot association. Pugh and Alcherio [42] present a simulated PSO-inspired collective search. In contrast to this thesis their research is focused on noisy fitness, dynamic environment, movement limitations and collision avoidance. As those factors are ignored here, their work is a very good resource on the problems not addressed in this thesis.

Some research also uses PSO within real robotic environments. Krishnanand and Ghose [26] conducted an experiment in which Glowworm Swarm Optimisation is performed by a swarm of Kinbots. The relative localisation is achieved by a special sensor built into the robots. Jatmiko et al. [17] successfully used PSO for odour source localisation with robots tracked by a camera.

2.1.2 Attraction Repulsion Behaviour

Another task that can be solved by swarm behaviour is formation control. Attraction repulsion behaviours are used to generate a certain pattern or distribution of robots in the environment. The interaction mechanism in attraction repulsion behaviours is an artificial force acting between the particles. An in depth overview for those algorithms is given by Gazi and Passino [13]. The behaviours can also be used for collision avoidance in a swarm of robots [12]. Attraction repulsion behaviours use a similar movement- and interaction model as PSO, thus PSO can easily be replaced by them. Another useful behaviour is the comfortable distance [13]. One application for this is keeping distances low enough for stable communication in the swarm. Attraction repulsion behaviours can also be used to implement leader following [13, 46].

2.2 Localisation Methods

The key to use swarm intelligence algorithms in robotic applications is the transformation of sensor data into the data needed by the algorithm. The most important component of the new algorithm is the localisation of the other robots in the swarm. Many localisation algorithms exist and most algorithms solve the localisation problem for a specific set of sensors. In outdoor use-cases GPS [15] (and alternatives like GLONASS) is the most widely used technology. Indoors, the GPS reception and accuracy are usually not good enough. This is the reason why most research on localisation is focused on indoor localisation. Hightower and Boriello [14] give and overview on those technologies.

The huge variety of sensors and environmental factors entail the need for many use-case-specific algorithms and technologies. Often, the environment is adapted for the robotic application. In indoor localisation no "default" solution such as GPS exists. In addition to the localisation itself, the orientation of the coordinate system and the robot must be known. Otherwise, the movement can not be executed by the robot. Localisation methods can be categorised by the following criteria.

Point of Observation A localisation system is called external, when the pose or important sensor data are transmitted to the robot from outside. It

is called internal when the robot computes its pose based on its own sensors.

Anchor Use A system that uses sensors, transmitters or receivers at known positions in the environment is called anchor-based. The fixed piece of equipment is called anchor (node).

Landmark Use Passive components in the environment are called landmarks, a typical landmark is an AR-tag (see [24]).

Motion If the nodes in the system do not move, the system is called static or motionless. If movement is required the algorithm is called motion-based.

Sensor Type The localisation algorithms can also be distinguished by the set of sensors that are used for localisation.

2.2.1 External Methods

The most commonly used technology for localisation is GPS [15]. It works well for outdoor applications, however, in indoors GPS is not available. Additionally, GPS is not accurate enough for some applications (i.e. collision avoidance for robots close to each other).

Another technology that is often used is camera tracking. In camera tracking systems, the robot is usually equipped with markers (i.e AR-Tags [24] or reflective markers [28]) that can be localised by processing the cameras' images. To obtain a 3D position and orientation usually multiple cameras are used. The flying machine arena at E.T.H. Zurich uses a very advanced setup using up to 20 cameras to track quadcopters [28]. In external camera tracking systems the images are processed and the final position estimate is transmitted to the robot [33, 24, 28].

While camera-based systems are often expensive and difficult to install and calibrate. They offer good localisation accuracy and update rates. Additionally, the hardware integrated into the robots using external localisation is minimal compared to the sensors needed for other localisation techniques. When a central component as the localisation is external to the robots, the control of the robots themselves can also be offloaded (as in [28]). Offloading the control of

the robots has the advantage that even less hardware must to be carried by the robots. However, a centralised system is less flexible, so other solutions are still important for use-cases in which flexibility matters.

2.2.2 Anchor- and Landmark-Based Localisation

Many indoor localisation systems are based on landmarks or anchor nodes. An anchor node is a piece of hardware that has a known position and the robots are capable to measure their distance (or angle) to the anchor node. Based on the range (or angle) measurements, the robot can use trilateration (or triangulation) to estimate its position. Landmarks are similar to anchor nodes as they are features at fixed locations in the environment. When the landmark positions are known, trilateration (or triangulation) can also be used to estimate the robot's position. Jayatilleke et al. [19] make use of AR-tags with known locations to localise the robot in the environment ('external camera tracking in reverse').

If the landmarks have an unknown position, a robot can still use them for navigation. Simultaneous Localisation and Mapping (SLAM) [40] is a method to create a map of the environment and to keep track of the robot's position within the map. SLAM is usually not used in multi-agent systems, as the intra-swarm localisation is not easily solved by SLAM. Still, the research by Pfingsthorn et al. [40] shows that it is possible to synchronise multiple maps to each other. When the maps of two robots are synchronised, relative localisation is possible.

SLAM is especially interesting for swarm intelligence algorithms that use stigmergy. The map of the environment can be used to store information about the virtual pheromone traces used in ACO. The disadvantage of using SLAM-based localisation is that SLAM is dependent on the environment. The environment must be static and contain enough features to generate a reliable map. Moreover, a representation of the map must be exchanged between the robots. This can be an obstacle to scalability in larger swarms.

2.2.3 Anchor-Free Localisation

Another class of localisation algorithms are anchor-free methods. In contrast to the anchor-based methods, no anchors are placed in the environment, but the localisation information is just drawn from measurements made between the robots. Usually, anchor-free localisation methods compute a relative rather than an absolute position as there is no reference point in the environment. However, anchor nodes can still be used to compute a global position from the relative position of the anchor.

One approach to localise nodes is to analyse distance measurements between the nodes. When a graph with node distances is given, the position of three points can be used as a reference to localise a fourth node [34]. The Self Positioning Algorithm by Capkun et al. [5] uses a similar method. A point is used as origin of the coordinate system, while two other points are used as a reference to locate all other points. This method was improved by clustering the nodes to reduce communication overhead (CSPA) [16]. The more recent Matrix-Transform Based SPA (MSPA) [53] uses matrix transformations to join the locally established coordinate systems.

The advantage of the SPA algorithms is that only distance measurements are needed for relative estimation. The main disadvantage is the lack of orientation information. While the relative position of the particles is needed for swarm algorithms, an important information is still missing: The orientation of the coordinate system. Without knowing the orientation of the robot within the coordinate system, a new movement can be computed, but it can not be translated into an unambiguous movement command.

This problem could be mitigated, if either anchor nodes are present to link the relative coordinates to a global reference, or the orientation can be measured.

The algorithm chosen for the new method is based on the GPS-free Directed Localization algorithm by Akcan et al. [2]. GDL was developed for applications in the area of Wireless Sensor Networks. The advantage of the algorithm is that it is based on the movement vectors of the particles and on distance measurements. The algorithm stands out, as it is motion-based while most other algorithms are designed for motionless networks.

The algorithm works by computing relative position estimates for each pair of particles within a fixed communication radius by solving a similar geometric problem as described in the next chapter (subsection 3.5.1). The algorithm is able to deduce the relative location of two points, given the velocity vectors between two time steps and the distances at those time steps. Because of symmetry, two solutions to the equations exist, of which only one is correct. The correct solution is then determined by comparing the position estimates of two neighbouring particles with the distance between the particles to filter the correct solution. The GDL algorithm uses only the knowledge gathered in the last time step and filters solutions that are expected to have high error or can not be computed because of singularities in the equations. This results in a very undesirable property of the original GDL algorithm: A location estimate is not yielded in some of the time steps. Additionally, a robot using GDL must be able to measure its distance to other robots and its movement direction and speed, restricting the possible applications for the algorithm [2].

As the particles must be able to execute movement commands for PSO, measuring movement speed and direction is a key aspect of the algorithm, even without the requirement of the localisation algorithm. A variety of sensors is also available to measure inter-particle distances. Also, the mathematical expression of the GDL algorithm is very close to the expression of swarm intelligence algorithms. Both algorithms, GDL [2] and PSO[56], rely on directed movement happening in discrete time steps. While the swarm movement is usually computed within absolute coordinates, GDL computes local coordinates. Reformulating PSO to use local coordinates is simple at least for particle to particle interaction.

Two algorithms have been developed by Akcan et al. following the idea of GDL. The GPS- and Compass free Directional Localization algorithm GCDL [3] is able to localise the nodes without the need of a compass by moving the nodes in a specific pattern. Dual Wireless Radio Localisation DWRL [1] is less similar to the original GDL. It uses two radio modules to obtain four distance measurements for each pair of nodes. Both GCDL and DWRL could lead to another version of the localisation algorithm that does not require the knowledge of a shared reference direction.

2.3 Measurements

Multiple sensors can measure the values required by the GDL algorithm [2]. This section shows that the sensors required to run the algorithm exist in the real world. The inputs for the GDL algorithm are a movement vector for each particle and the distances between the particles. The movement consists of two entangled pieces of information: Movement distance and direction [2]. The PSO algorithm also uses the fitness of the particle at the current position. The objective is specific to each application, so no such sensor can be examined here.

2.3.1 Movement

One measurement needed in the localisation algorithm is the movement in each time step. Both covered distance and movement direction need to be measured, with respect to a global frame of reference.

Movement is often measured where it is generated: At the wheels [38, 6]. Sometimes this is called odometry: "Odometry is the measurement of wheel rotation as a function of time" [6]. However, the term odometry is often used for other methods of measuring movement as well (i.e. "visual odometry" [37]). By measuring the turn rate of the wheels the speed of a robot can be determined. Applying the method in robotic applications entails problems, though. When the robot does not move in a straight line, a complex model is needed to compute the trajectory of the robot. Additionally, slipping wheels and rough ground can distort the measurements. Chong and Kleeman [6] give an overview about the error sources of odometry measurements in two wheeled robots.

Other methods to measure speed do not involve the actuators. In visual odometry, camera images are processed to generate movement data [37]. A sensor specialised in measuring movement over a surface can be found in an computer mouse[1]. Most computer mouses work by measuring optical flow. Bees use optical flow to navigate, too [51]. By measuring how fast the image of a camera moves, an optical flow sensor can determine velocity in two dimensions and measure covered distance. The main difficulties of optical flow sensors are to

[1] While most computers use an optical mouse nowadays, wheel odometry was also used in the past.

find features in the environment to track in the camera image, and to measure the distance between those features and the camera. This means the robot has to move slowly enough, so the frames overlap enough to track the features. Essentially, there is an upper bound robot speed that is measurable by sensors of this type. Additional problems exist if the camera is not mounted within a fixed distance and orientation to the ground, i.e. in flying robots. One problem that comes with changing the distance to the ground is the changing focal pane of the camera. The image quality is reduced when the focus is not set according to the distance. Severe errors can occur when a flying robot rotates around the pitch- or yaw axis. A pitch motion introduces a fast optical flow that does not correspond to robot motion. It can exceed the inherent speed limit where the camera frames do not overlap any more [48].

Some of the landmark-based techniques for localisation can also be used for measuring the robot's movement in space. For example, the movement of a robot can be tracked with the SLAM method [40]. The position of the robot in the map can be compared to an old position to obtain the movement that occurred in between the two positions. While SLAM usually operates in an unprepared environment, specially placed landmarks can also be tracked optically to deduce the robot's movement with reference to the landmark. In case enough landmarks are placed in the robot's environment, they can be used to compute robot movement reliably [19]. Usually, this method is used to compute an absolute position of the robot, but for absolute positioning the position and orientation of the landmarks must be known. However, landmark-based localisation schemes such as [19, 24] can be used to compute relative movement in relation to fixed points with unknown location to deduce movement speed. Most of the presented methods measure more than speed: The goal of all methods is to provide information on the covered path of a robot. The localisation obtained by following the robot's movement from a known position onwards is called dead-reckoning [23, 11]. Dead-reckoning is often used to improve the quality of absolute localisation methods such as GPS [23, 21].

A key piece of information that is often neglected is the orientation of the robot. Methods to determine a robot's orientation that are used outdoors are hardly usable in indoor environments. Those methods include compass-like magnetometer measurements and estimating the movement direction by tracking the GPS position over time. While GPS is not receivable in indoor

environments, the magnetic field is often not consistent indoors. Still, modified versions of some outdoor methods can often be applied indoors.

One possibility to find out the robot's orientation is celestial navigation [52]. When the orientation towards a far away landmark like the sun or the stars can be measured, a globally shared orientation can be deduced. In indoor environments such landmarks may exist (i.e. a single light in the room) or can be built (i.e. IR-beacons). The Kinbots use an IR sensor [26] to measure the angle of arrival of an IR signal that could be suitable for heading estimation in conjunction with a landmark. Furthermore, some landmarks may directly offer orientation information [24].

All in all, the movement direction in a shared frame of reference is difficult to obtain. A localisation method that does not need this information for heading estimation is therefore desirable. Tough, some of the localisation schemes do not provide the robots with an information on their own orientation within the coordinate system. Hence, the robots still need a sensor measuring their orientation.

2.3.2 Distance

Measuring distance is often necessary in robotic applications. Two different types of distance measurement exist. Some sensors measure their distance to the first object within the sensor's measurement area. Examples for those sensors are IR distance sensors or sonar distance sensors. The type of distance sensor used in localisation algorithms (like trilateration) needs to measure the distance from one distance sensor to another specific sensor. Usually, both sensors are of the same type and an address is used to identify which sensors are measuring their distance [47, 50]. Some systems also allow broadcast measurements, where one sensor broadcasts a signal and all other sensors can measure their distance to the beacon that started the measurement [50, 36].

Signal Strength

A very common method to determine distances is measuring signal strength [55, 36]. As receiver and transmitter of a wireless signal move away from each other, the received signal strength decays. This mechanism can be

used to measure distances on already existing wireless communication platforms, e.g. WLAN, Bluetooth or ZigBee. A major problem of the method is the propagation pattern of the signal, that is dependent on the antenna type and orientation. The orientation of the transmitter and receiver antenna always matters. This introduces a really strong error into the measurements. Additionally, the environment matters. Wireless signals can be reflected or absorbed by different materials, therefore, the measurement is strongly dependent on the environment [55, 36].

Time of Arrival

A mechanism that can also be used to measure distances is the time a signal needs to travel from transmitter to receiver (Time of Arrival, ToA). As the signal in wireless communication usually travels at light speed, the time intervals that must be measured are very short. Nevertheless, using the propagation time instead of signal strength has the benefit that antenna direction is not an important factor to the measurement any more [47, 22]. The environment is still influential on the measurement. The signal can be reflected in the environment if the reflection is measured instead of the real signal. In case this happens the measured range is distorted. This source of error is called multipath effect [47].

There are multiple ways to measure the time the signal needs to travel. In GPS the clocks are synchronised and message is transmitted, that contains the transmission time stamp. The time signal that is transmitted by the satellites can be used to synchronise the clock of the receiver to the GPS network. Once synchronised, the travel time of the signal can be measured. With an unsynchronised system such as the Decawave sensor, round trips with receive and transmit time stamps are used to determine the travelled time by factoring the time needed for returning the signal out of the round trip time [20].

Time Difference of Arrival

A way to measure distances, without the real time constraints of measuring speed-of-light signal propagation time, is to change the medium the signal travels in. The cricket [50] and active-bat [54] systems use ultrasound pulses to

measure the travel time. Because sound is much slower than light, measuring the time spans accurately is easier. But using ultrasound has another advantage. An ultrasound signal and a RF-signal are transmitted simultaneously and the time difference of arrival (TDoA) is measured. The speed difference of the signals is known, therefore, the distance can be computed from the TDoA.

A major drawback of the slow travel time of the signal is crosstalk between the nodes. When one node sends an ultrasound signal, the signal must fade before another signal is issued by another node sharing the medium. If the signal is not faded before the next signal is sent, the receiver can mistake both signals and measure the wrong distance. This leads to a limit to the measurement frequency in the hole network (or at least sub-networks). As crosstalk is a problem, cricket or active-bat are less suited for large networks with high update rates. Advanced scheduling can mitigate the problem, though [54].

Phase Difference

A feature of RF-signals that can be used to measure a distance is the phase of the signal. By measuring the phase of the signal on multiple frequencies, the distance between the nodes can be determined. A system that uses this type of ranging is the ATMEL RTB [49, 29].

Other Methods

One way to compute distances is to use camera tracking. It is possible to recognise other robots in an image and the size of the robot in the image can be used to determine its distance. A swarm method to measure distances is the Hop Count method. However, Hop Count is not well suited for swarms with small densities [32]. Additionally, the hop count method is not suited to measure the distance of direct neighbours. Other methods for intra-swarm distance estimation such as Distributed geometric distance estimation [31] can mitigate this problem.

Many methods exist to combine multiple sensors readings into a single information and to improve a single sensor reading by an underlying model. Those techniques are not discussed in this thesis.

Standard PSO 2011 [56] was chosen as the swarm intelligence algorithm to be used. It can easily be replaced by other PSO-inspired algorithms and attraction repulsion behaviours. The PSO will be modified to work with local position estimates that are generated by a new localisation algorithm based on GDL [2]. Dead-reckoning can be used to overcome the disadvantage of GDL not yielding an estimate in each time step. Following the name of Simultaneous Localisation and Mapping, the new algorithm is called Simultaneous Localisation and Optimisation (SLO).

How the new algorithm works is described in the next chapter.

3 Simultaneous Localisation and Optimisation Algorithm

The algorithm created for this thesis is called Simultaneous Localisation and Optimisation method. The aim of the SLO method is to combine two major components: A localisation algorithm inspired by the GDL algorithm by Akcan et al. [2] and SPSO 2011 by Zambrano-Bigiarini et al. [56]. The idea behind this combination is to create a swarm algorithm for a robot that perceives the environment through sensors and acts on it through actors. Both algorithms must be modified to create one functioning unit. The original GDL algorithm does not yield a position estimate in every time step. To overcome this disadvantage the new algorithm uses dead-reckoning to close the gaps. SPSO must be modified to work on local coordinates, as GDL does not compute global coordinates.

3.1 Definition of the Environment

All particles move within a continuous two dimensional plane without obstacles. Moreover, an objective function maps a fitness value to each point in the plane. In this thesis, fitness is minimised in all objectives. The goal of the particles is to find the global minimum of the objective function on the plane.

All information on the environment is gathered through sensor measurements and changes in the environment are caused by actors. Each measurement can contain an error. The position of the other robots must be perceived through sensors too. Movement is an action of a particle within the sensor-actor model. The particles are assumed to move perfectly according to the movement command. This simplification is made to have fewer entangled

effects[1]. The feedback of the action is still infused with an error through the sensor, so the error-free movement is not an oversimplification.

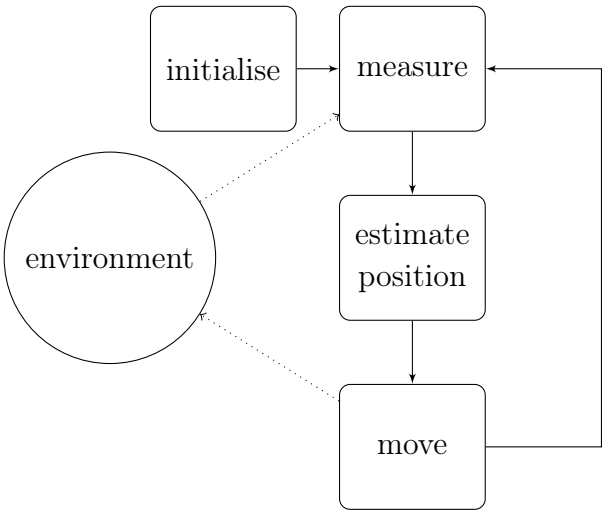


Figure 3.1: Overview of the SLO Algorithm and its Interaction with the Environment.

A very simple model of timing is used. Time passes in discrete steps. The measurements made by the sensors are assumed to happen without delay, as well as the movement of the particles. All particles perform their behaviour in each time step. A consequence of the simple model of time is that synchronous updates do not suffer from any drawbacks synchronisation would usually have in a robotic environment. Examples for those drawbacks are increased time delays and communication overhead.

3.2 Overview

The SLO method combines two major components: A localisation algorithm inspired by the GDL algorithm by Akcan et al. [2] and a modified version of SPSO 2011 by Zambrano-Bigiarini et al [56]. Figure 3.1 shows how the algorithm works and how it interacts with the environment. First, all the variables in the algorithm are initialised. After initialisation, the algorithm runs in a loop until a stopping criterion is reached. In this loop three major things happen. (1) The sensors measure the data in the environment. (2) The

[1] An imperfect movement has been implemented, but was not part of the experiment.

relative positions of the particles are estimated. (3) The particles act according to the rules of swarm intelligence.

In Algorithm 1, a more detailed description of the algorithm is given. In the initialisation phase, the population of size P is initialised with values randomly placed in the search space S. Additionally, the estimates and velocities of the "previous" time step are initialised with random values.

After the initialisation the main loop of the algorithm begins. In each time step the sensors measure their values. The measurements include the neighbourhood relationship N, the distances of the particles to each other D and the velocity vectors of the particles V. The previous estimate and the measurements are used to compute the new position estimate $\vec{x}_i^*(t)$. However, the core localisation algorithm yields two symmetric solutions X. Therefore, an additional step is needed to select the right position estimate from those solutions. To guide this selection a third estimate obtained by dead-reckoning is included in X, too. The last step of the algorithm is the position update. Updating the position includes the computation of the desired movement $v_i(t+1)$ for each particle with PSO and the actual update of the particle positions in the environment. In addition, the previous best value of the PSO must be updated, because the coordinate system is moved when the particle is moved. All steps of the algorithm are explained in more detail in the next sections.

Algorithm 1: SLO-PSO algorithm

Input : P, S
t = 0
Initialise the PSO individuals:
for $i = 1$ *to* P **do**
  $\vec{x}_i(t)$ = RandomPosition(S)
  $\vec{v}_i(t)$ = RandomMovement()
  $\vec{x}_i^*$ = RandomPosition(S)
end
while *stopping criterion not fulfilled* **do**
  for $i = 1$ *to* P **do**
    `/* Measurement:` `*/`
    N_i = UpdateNeighbours(i)
    D = MeasureDistances(N_i, i)
    V = MeasureVelocity()
    `/* Position Estimation:` `*/`
    X = ComputeLocalisationOptions($V, D, \vec{x}_i^*(t-1)$)
    $\vec{x}_i^*(t)$ = SelectSolution(X, D, i)
    `/* Movement:` `*/`
    $\vec{x}_g^*(t)$ = FindGlobalBest($N_i, \vec{x}_i^*(t), i$)
    $\vec{v}_i(t+1)$ = ComputeVelocity ($\vec{x}_i(t)$, $\vec{x}_g^*(t)$)
    $\vec{x}_i(t+1)$ = UpdatePosition($\vec{v}_i(t)$, $\vec{x}_i(t)$)
  end
  $t = t + 1$
end

3.3 Initialisation

In the initialisation phase multiple variables are set up. The particle positions are chosen. In a robotic application the robots must be placed in the environment. As the algorithm uses data from previous time steps the variables holding this data are initialised randomly. The estimates for the previous position of the particles are randomly placed within the search space. The velocities are the difference vectors between those arrays and another set of points randomly placed in the search space. In a robotic application this step

may be more complex i.e. by creating a known initial state, or a good estimate for the initial state.

3.4 Measurement Phase

The idea behind SLO is to replace direct knowledge of the environment with sensor-deduced information. Multiple sensors are required to use the algorithm. The simulation runs with discrete time steps. In each step the following values must be measured: The movement in between the last and the current time step ($\vec{v}_t^*$), the distances between the particles in the current time step ($d_t^*(A, B)$) and the fitness of each particle at its current position ($f(x_i(t))$). The movement measurement needs to include the direction of movement, as well as the movement speed and is represented as velocity vector.

Additionally, the estimates of the last time step (x_{t-1}^*) and the distances measured in the last time step ($d_{t-1}^*(A, B)$) are used in the algorithm.

The neighbourhood of the particles may change because of the particle movement. The neighbourhood relationships are strictly based on the real particle positions from the simulation. Therefore, the neighbourhood update is a measurement. The following neighbourhoods are considered in this thesis:

k-nearest-neighbours (knn) The k points closest to a point A are its neighbours.

communication radius All points B with $d(A, B) < r$ are neighbours of point A.

fully-connected Each point is the neighbour of each other point.

random At each time step point B is determined to be a neighbour of point A with probability p.

ring Each point X_i has the neighbours $X_{i \pm k mod N} \forall k \leq n$, where n controls the connectedness of the graph (the bigger n, the more connections).

The fully-connected, knn and communication radius neighbourhoods are the most realistic neighbourhoods in a robotic application. However, the knn and communication radius combine different properties of a neighbourhood that are interesting to analyse. The interesting properties are *connectedness* and *rate of*

change. The degree to which the particles are connected and how many changes happen in the neighbourhood can be controlled either indirectly or not at all for the more realistic neighbourhoods. Additionally, those neighbourhoods are distance-based and affect the absolute localisation error by filtering the particles for their distances. The random topology and ring topology were added to address those issues. The ring topology was first introduced as a comparison to the fully-connected network. Both are static topologies with a different degree of connectedness. However, the knn and communication radius topology already include a parameter that controls how densely the network is connected. For that reason the parameter n was added to the ring topology that allows to include more particles in the neighbourhood based on the index of the particles. To control the rate of change in the neighbourhood, the random topology was added. The parameter p defines a probability that decides whether a connection in the neighbourhood graph exists in each time step. While p controls the rate of change as well as the connectedness, the random topology has the advantage that the neighbourhood relationship is agnostic to the distance of the particles. Therefore, p does not affect the absolute localisation error in the same way knn and communication radius topologies do.

3.5 Localisation

After the measurements are aggregated, a position estimate for each particle must be computed. The position estimate will later be used by the PSO algorithm to compute a movement command. Computing the position estimates is a two-step process. First, three solution candidates are computed. In the second step the correct solution is chosen.

3.5.1 Localisation: Solution Computation

The localisation scheme in the SLO method is based on the GDL algorithm by Akcan et al. [2]. In GDL, the solution to the geometrical problem yields two solutions and the correct solution is determined in a later step. As the SLO localisation is based on GDL, the same symmetry exists and the localisation yields two solutions. In contrast to GDL a third solution is used in SLO. The

third solution is computed by dead-reckoning and is not accurate if the last estimate is not correct. Since the dead-reckoning solution is computed in a different way, it can help to decide on the correct solution.

To properly explain the computation the position estimate is assumed to be computed from A to point B. An estimate for the value of the vector $\vec{AB}$ is the result of the computation as it represents the relative position of point B with respect to the reference point A. Furthermore, point A is assumed to be at $(0, 0)$ and only point B moves with the combined velocities of A and B: $\vec{v}^* = \vec{v}^*_B - \vec{v}^*_A$. Both directions of the estimate are computed independently ($\vec{BA}$ is computed in another call of the function ComputeLocalisationOptions as $\vec{AB}$).

As mentioned, two approaches are used to compute an estimate for $\vec{AB}$. The easiest way to obtain an estimate is to update the old position with the velocity vector: $\vec{AB}_t = \vec{AB}_{t-1} + \vec{v}^*$. This solution is called $\vec{x}^*_3$ in the next steps of the algorithm. However, $\vec{x}^*_3$ is not sufficient to solve the localisation problem, as $\vec{AB}_{t-1}$ is not initially known and $\vec{v}^*$ is a value that contains measurement errors. Another method to compute $\vec{AB}_t$ is needed to solve the initialisation problem and prevent the accumulation of measurement errors.

The second method to estimate $\vec{x}^*_{1,2} = \vec{AB}$ is the method inspired by the GDL algorithm by Akcan et al [2]. When the distance $d(A, B)$ is measured at each time step the points A, B_t, B_{t-1} form a triangle with sides of known lengths as shown in Figure 3.2. Therefore, Equation 3.1 can be used to compute the angle α. The angle ϕ can be computed with Equation 3.2. This is possible when the movement vector can be measured in a common frame of reference. With α, ϕ and the distance $d(A, B_t)$ an estimate of $\vec{AB}$ can be computed as shown in Equation 3.3.

The major modification of the GDL algorithm [2] is the different formulation of the problem. By assuming point A to be at $(0, 0)$ at all time steps, the formulas become simpler and the geometric problem becomes easier to analyse and understand. The underlying mathematical problem is not changed by this assumption as just the coordinate system is moved.

This method has a disadvantage, though. As $cos(\alpha)$ is computed, α is derived by computing an inverse cosine. However, *arccos* yields two valid, non equivalent solutions: $+\alpha$ and $-\alpha$. Geometrically, both solutions exist because of symmetry as shown in Figure 3.3. As one of the solutions is an artefact and

one is the real solution to the localisation problem, a method is needed to find the correct one out of the three solutions $\{\vec{x}_1^*, \vec{x}_2^*, \vec{x}_3^*\}$.

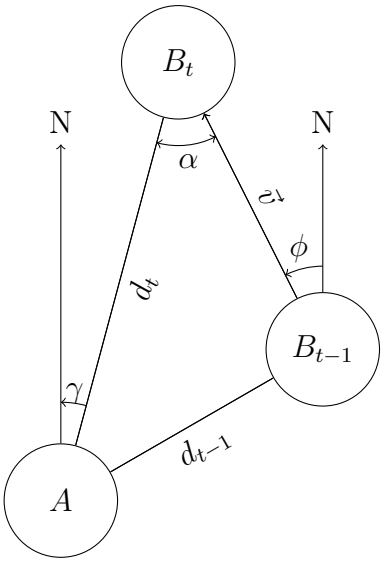


Figure 3.2: Triangle Used for Position Estimation. The triangle that the points A, B_t, B_{t-1} form, labelled with the quantified needed for the equations 3.1-3.3. $\vec{v}$ is the combined movement of A and B from the last time step ($\vec{v} = \vec{v}_B - \vec{v}_A$) and $\gamma = \phi \pm \alpha$.

$$cos(\alpha) = \frac{d_t^{*2} + |\vec{v}^*|^2 - d_{t-1}^{*2}}{2 \cdot |\vec{v}^*| \cdot d_{t-1}^*} \tag{3.1}$$

$$\phi = atan2(v_x^*, v_y^*) \tag{3.2}$$

$$(\vec{x}_1^*, \vec{x}_2^*) = \begin{pmatrix} sin(\phi \pm \alpha) \\ cos(\phi \pm \alpha) \end{pmatrix} \cdot d_t^* \tag{3.3}$$

$$\vec{x}_3^* = \vec{x}_{t-1}^* + \vec{v}_{t-1}^* \tag{3.4}$$

3.5.2 Error Mitigation

Some values in the estimation predictably amplify the errors of the estimation inputs. This is the reason why an intermediate computation step in the computation of the solution candidates can lower the localisation error. Two

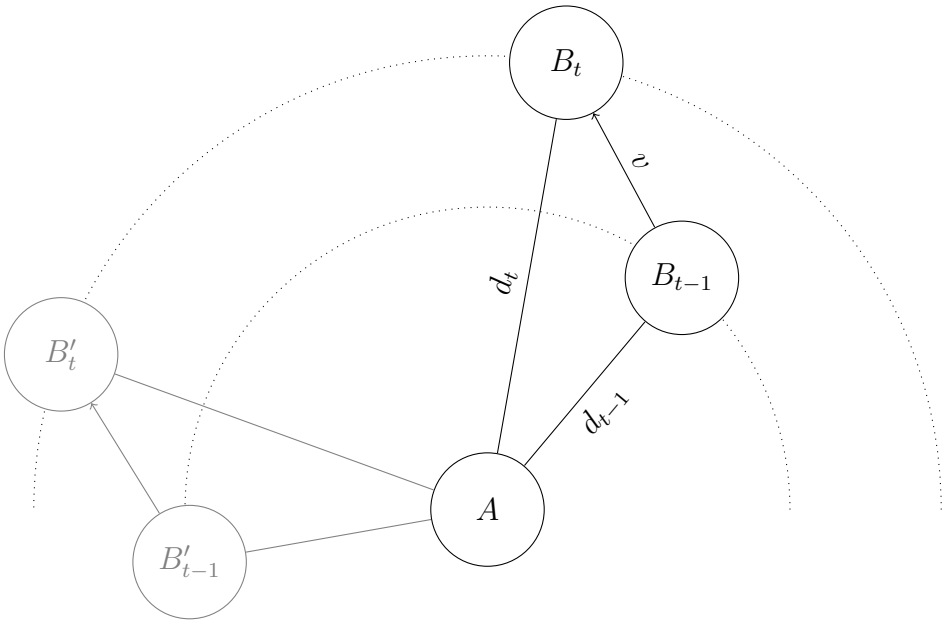


Figure 3.3: Symmetry of the solutions to the position estimation problem.

factors have been identified that predict error amplification in the two solutions of the geometrical problem.

First, the inversion of $cos(\alpha)$ in the computation of α can be inaccurate if $cos(\alpha)$ is close to one. The reason is: If $cos(\alpha)$ is close to one, measurement errors may result in a singularity where the computed value for $cos(\alpha)$ is not in the cosine range and therefore no value can be computed. The singularity problem can be solved in two ways: Clipping $cos(\alpha)$ to the range $[-1, 1]$ or deleting values for $\vec{x}^*_{1,2}$ that correspond to the singularity from the tuple of possible solutions. If $\vec{x}^*_{1,2}$ are deleted, $\vec{x}^*_3$ will be used as the estimate (the solution selection process is omitted). Equation 3.5 shows the criterion used to decide whether $\vec{x}^*_{1,2}$ should be deleted. If δ_α is smaller than zero, the condition is still applied and the value is clipped in the computation. This is the most general formulation of the problem, hence this solution has been implemented.[2]

The second variable that is used to estimate the localisation error is the magnitude of the combined velocity vector $|\vec{v}|$. When the particles stay still or move in parallel, $|\vec{v}^*|$ becomes small. Small values of $|\vec{v}^*|$ lead to the triangle in Figure 3.2 becoming very narrow. The ranging errors in d^*_t and d^*_{t-1} become

[2]As the experiments show, clipping the $cos(\alpha)$ to the cosine range is the preferred solution for future implementations.

more prominent compared to $|\vec{v}^*|$. The conditions used to detect this type of error are shown in Equation 3.6. Similar to the error mitigation with the cosine condition, the values for $\vec{x}_1^*$ and $\vec{x}_2^*$ are withheld from the selection method and $\vec{x}_3^*$ is used as the estimate in the current time step when the condition is true.

$$1 - |cos(\alpha)| \leq \delta_\alpha \tag{3.5}$$

$$\frac{|\vec{v}^*|}{d_{t-1}^*} \leq \delta_v \qquad \frac{|\vec{v}^*|}{d_t^*} \leq \delta_v \tag{3.6}$$

3.5.3 Localisation: Solution Selection

Closest Solution Selection

After the solution candidates are computed, a solution must be chosen to be used in the PSO.

One possibility to select a solution from the solution candidates is to compare the solutions generated by both methods. When the algorithm has previously found a good estimate for $\vec{x}_{t-1}^*$, the dead-reckoning solution $\vec{x}_3^*$ should be close to one of the two solutions obtained by the distance-based estimation ($\vec{x}_{1,2}^*$). Therefore, a weighted average between $\vec{x}_3^*$ and the other solution closer to it ($\vec{x}_c^*$) as shown in Equation 3.7 is a good estimate for the next position. The advantage of using a weighted average is that the trade-off between the errors in both estimation methods can be fine-tuned to the use-case of the algorithm.

$$\vec{x}_t^* = w\ (\vec{x}_{t-1}^* + \vec{v}^*) + (1 - w)\ \vec{v}_c^* \tag{3.7}$$

The closest solution selection is the least complex of the three selection methods implemented. While the computational power needed is very low and there is very little communication, the algorithm has cold start properties: When the initial position of the particles is unknown, the algorithm struggles to compute the next solution. If the algorithm runs, the estimate should converge to the real solution, so the computation is more accurate once a good solution has been found. This is especially important when the neighbourhood of the particles is likely to change. Unless there is a strategy to localise new particles, the algorithm takes multiple time steps to converge to a good solution. If

the neighbourhood changes too frequently, this algorithm only produces poor results.

Round Trip Selection

$$||\vec{AB} + \vec{BC} + \vec{CA}|| = ||\vec{AA}|| + e_r = 0 + e_r \tag{3.8}$$

The selection of a solution can also be based on a criterion for the relationship of the particles in the swarm. The Round Trip Selection Method (RTS) uses the relationship shown in Equation 3.9. When the estimates of a three way round trip from A to B to C are added, the result should by definition be 0, as it is equal to the path from A to A. This means the solution candidates for $\vec{AB}$, $\vec{BC}$, $\vec{CA}$ can be picked to minimise the round trip error e_r in Equation 3.9. That way all the solution sets can be reduced to single solutions unless no triangles are found in the neighbourhood. In that case the CSS can act as a fallback (if only a single point is in the neighbourhood, or no triangle can be formed).

While in the first implementation [30] the best triangle was used first and the backward edges were adjusted as soon as the forward edge was determined, the order of computation has changed in the current implementation. As the new implementation allows to comply with neighbourhood relationships not every edge has a corresponding backward edge. Additionally, there is a possibility that no three way round trip including a certain edge exists (i.e. in a ring topology). Because of this the new implementation of the algorithm uses the first viable triangle and reduces just a single edge. Reducing multiple possibilities at once is not possible any more. The first triangle used can be the wrong choice, thus correct edges that are needed to solve later round trips correctly are removed. Finding the optimal set of solutions according to the triangle metric is a np-hard optimisation problem in itself. The current implementation only uses a simple greedy approach to obtain a suboptimal solution. Investing more computational power potentially unlocks a more accurate localisation algorithm.

There is a problem with the RTS selection method and with the current implementation it is more likely to occur. The metric checks the estimates for consistency, not for correctness. Initially, the more consistent solution is likely to be the correct one. The solutions obtained by dead-reckoning are always

equally consistent with each other as the previous ones. If there is an error in the velocity, the new estimate contains an error which the consistency-metric can not account for. While this problem is severe in theory, it can be mitigated by changing the evaluation order.

Akcan's method

$$|(||\vec{AB} - \vec{AC}||) - d_t^*(B,C)| = ||\vec{AB} - \vec{AC}|| - ||\vec{BC}|| = 0 + e_d \tag{3.9}$$

Another metric to determine which of the two ranging solutions should be used, was used in the GDL algorithm by Akcan et al. [2]. As the distance between points is measured it can be compared to the distance of the estimates. The solution of the position estimation process where $||\vec{AB} - \vec{AC}|| - d_t^*(B,C)$ is minimised is chosen as the estimate in the current time step. Akcan et al. explicitly exclude knowledge about the past from their process, however, in this implementation the third solution $\vec{x}_3^*$ is also tested and used when it outperforms the other solutions.

In terms of communication overhead, computational cost and performance this method is in between the CSS and RTS method.

3.6 Movement Update

While the position estimation is a static part of the algorithm, the actual swarm behaviour can be considered as exchangeable. The swarm behaviour used in this thesis is based on the Standard Particle Swarm Optimisation algorithm (SPSO 2011) [56].

As in SPSO 2011 a point $G_i(t)$ is computed for each individual from the best particle in the neighbourhood $L_i(t)$ and the previous best position $P_i(t)$ (Equation 3.10[3]). While in the original SPSO 2011 algorithm the points P_i and L_i are both selected from past time steps, in this implementation only P_i uses information on past states, as the position information in a robotic environment can deteriorate as new measurement errors accumulate. $G_i(t)$ is the centre of

[3]As a local coordinate system is used terms including $X_i(t)$ get simplified as $X_i(t) = 0 \forall t$.

a hypersphere towards which the particle moves. The weights c_1 (cognitive term) and c_2 (social term) are used to scale the impact of P_i and L_i. c_3 and c_4 are added to tune the exploration behaviour. $c_3 = 1$ and $c_4 = 0$ represent the original behaviour of the algorithm. Reducing the area that the algorithm can reach by random choice could be beneficial to compensate for random errors in the localisation algorithm.

$$G_i(t) = 1/3\,(c_1 U_1 \otimes P_i(t) + c_2 U_2 \otimes L_i(t)) \tag{3.10}$$

$$V_i(t+1) = \omega V_i(t) + \mathcal{H}_i(G_i, c_3 \parallel G_i \parallel + c_4) \tag{3.11}$$

$$X_i(t+1) = V_i(t+1) \tag{3.12}$$

After the computation of the PSO, the particles' positions are updated according to the movement vectors computed by the swarm behaviour in Equation 3.11. Since all the particles use local coordinate systems, their knowledge about their environment needs to be updated as they are moving. This means that the information on the previous best particle position $P_i(t)$ must be updated, not only by checking whether a better fitness value has been observed, but also by correcting an old position by the movement of the particle as shown in Equation 3.13.

$$P_i(t+1) = P_i(t) - V_i(t+1) \tag{3.13}$$

4 Evaluation

The goals of this thesis include the implementation of the SLO algorithm and the analysis of its performance in an experiment. The algorithm is described in the previous chapter. This chapter describes how the algorithm was implemented and what experiments were performed. The results of the experiments are discussed as well.

4.1 Implementation of the Algorithm

The algorithm and the experiments are implemented using a Jupyter notebook. The implementation makes use of the composition software design pattern. The main loop of the algorithm is implemented, so the internal functions and arguments are passed as keyword arguments. Hence, the algorithm is composed of different parts that can be chosen independently.

A dictionary holds the data describing the algorithm run. The dictionary contains the function generating the simulated measurements, the position estimation function and the movement update function. Additionally, the dictionary contains the arguments to those functions, e.g. measurement errors, estimation parameters and PSO parameters. Some variables that control the algorithm run including initial population and the number of generations to be run are also included in the dictionary. All of the selection methods described in subsection 3.5.3 were implemented. The CSS selection method can be modified by the weight of the updated velocity w. Moreover, a perfect localisation is implemented as a baseline. However, the experiments have shown that the localisation algorithm (especially with RTS/Akcan's selection method) produces nearly the same result when no input error is used. So, the perfect localisation is not needed.

The output of one algorithm run is the true position of the swarm particles and all the position estimates. From the positions the particle fitness can be

reconstructed. The positions together with the estimates are used to compute the localisation error.

Most parts of the algorithm's functions are self-implemented. Some python modules implementing PSO exist, but usually those packages implement different algorithms to obtain the solution of an optimisation problem. Intermediate steps can not be modified by the user in those implementations, so they are not suited for this thesis. In the implementation the particle positions and movement vectors are represented as numpy array. Many operations can be executed by linear algebra functions provided by numpy (i.e. the movement update is formulated as matrix operation that is handled by numpy). The fitness functions used for benchmarking the algorithm are provided by the DEAP [7] python package.

To run and store the experiments the pypet-, and for handling data the pandas package is used. As the algorithm can be configured by a huge set of parameters, an automated way of running the experiment with different parameter settings is needed. This functionality is provided by the pypet package. For each parameter set the algorithm is run multiple times and the data computed in the experiment run are saved together with the set of parameters that was used. As the amount of data created by the algorithm itself is very high, only a part of the outcome can be saved (or otherwise the data would not fit into RAM in the evaluation phase). For each run in each parameter set the aggregated statistics for the localisation error (mean, standard deviation, median) for different types of error (absolute, relative, angular) is exported. Furthermore, the inter-particle distance is measured (mean, standard deviation, median). In the PSO experiments some additional values were exported that are described in section 4.6.

For evaluation the saved data that was saved are loaded and analysed. To create plots and handle statistical data the matplotlib and seaborn packages were used.

4.2 Parameter Space

The main difficulty in finding out how the algorithm behaves is that there are many parameters that affect the algorithm's performance. Therefore, the

method in the experiments is to fix most parameters to see how the modification of single parameters affects the overall result. Table 4.1 shows a summary of the parameters that are used for most experiments (in some of the experiments the error is divided into more components).

To reduce the amount of computation time, the swarm movement is replaced by a random walk movement in the first experiments. Later experiments use a fixed set of error- and mitigation parameters to better understand the interaction between PSO and localisation. Each experiment consists of several runs with different parameter sets. The runs themselves are repeated 31 times to generate statistically meaningful results. For each experiment one particular group of parameters is varied systematically to obtain data on the influence of individual parameters.

	Name	Default value
Algorithm	Selection Method	
	Prediction Weight	0.5
Neighbourhood	Function	knn
	Parameter	5
SPSO 2011	cognitive c_1	$0.5 + ln(2)$
	social c_2	$0.5 + ln(2)$
	c_3	1
	c_4	0
Random Walk	speed	5
Objective	Objective Function	line
Error Set		Low
Mitigation	δ_{cos}	0.03
	δ_v	0.03

Table 4.1: Parameters used in the experiment

4.3 Experiments

As mentioned before, testing all parameter combinations is not possible with the computational resources available. For this reason four experiments have been designed that change only a subset of the parameters while keeping most of the parameters fixed. The idea behind grouping the parameters is to find out the parameters' individual effect and to see where parameters interact.

The first experiments do not use the PSO movement to gain a better understanding of localisation accuracy. This is necessary as PSO and the large set of possible objectives would increase the size of the parameter space more than computing power allows. Only the last experiment looks into the effects of PSO movement. One reason why PSO is only handled in a single experiment is the vastly larger parameter space. Another is the fact that the more complex interactions between movement-based localisation and localisation based-movement often do not allow a conclusive interpretation.

The current implementation for the experiments works in two steps: First, the simulation is run and the data are exported to a file summarising the experiment's results and the parameters that were used. Later, the file is read and the results are analysed. The advantage of this is that the simulation can be run on a server, while the results can be viewed on a laptop. A disadvantage of the method is that only data that are saved in the simulation run are accessible when the results are evaluated.

The exported data are summarised in Table 4.2. For the localisation error, each generation mean, median and standard error are saved. The angle error is the difference between estimated direction of the particle and the real direction. The absolute error is the distance between estimate and real position of the other particle. In contrast to that, the relative error is the absolute particle divided by the distance between both particles.

Experiment	Variable Name	per Run
Random Walk and PSO	Mean Angle Error	
	Std Dev Angle Error	
	Median Angle Error	
	Mean Absolute Error	
	Std Dev Absolute Error	
	Median Absolute Error	
	Mean Relative Error	
	Std Dev Relative Error	
	Median Relative Error	
PSO only	Best Fitness	
	Best Fitness	x
	Fitness at Last Generation	x
	Generation Where Criterion is Met	x

Table 4.2: Data exported from the simulation run. The last column shows whether the data item is aggregated per generation or summarised per run.

4.4 Error Model

To gain knowledge about the impact of errors on the localisation performance, an error model was implemented that models several error types for both range and velocity measurements. Based on the error classification by Zug [58] several error types have been modelled.

The distance measurement ($d^*(t)$) is generated from the real value ($d(t)$) that is modified by multiplying (E_m) and adding (E_a) an error value (Equation 4.1). E_m simulates value correlated errors, while E_a simulates constant errors. E_a is computed by adding two normally distributed values (measurement noise is often assumed to be normally distributed [58, p. 21]). One of the values is drawn at the beginning of the run ($P = \mathcal{N}(0, \hat{\sigma})$). This value models permanent errors, while the other value is drawn each generation to model stochastic errors (Equation 4.3).

Additionally, the measured value is set to zero with a probability p_{fail} to emulate (filtered) outliers, omissions and stuck-at-zero errors without drastically increasing parameter space. Time correlated errors and delays have not been modelled as timing related issues are simplified by the discrete time model anyway.

$$d^*(t) = \begin{cases} 0 & \text{sensor fails } (p_{fail}) \\ d(t) \cdot E_m + E_a & \text{else} \end{cases} \tag{4.1}$$

$$E_m = \mathcal{N}(1, \sigma_m) + P_m \tag{4.2}$$

$$E_a = \mathcal{N}(0, \sigma_a) + P_a \tag{4.3}$$

$$\vec{v}^*(t) = R(\mathcal{N}(0, \sigma_r) + P_r)\vec{v}'(t) \tag{4.4}$$

The movement measurement $\vec{v}^*(t)$ is computed similarly to the distance measurement. E_a and E_m are individually computed and applied for each dimension, while the sensor failures p_{fail} affect the whole vector. This value ($\vec{v}'(t)$) is then subject to a rotation with a normally distributed angle (Equation 4.4). Similar to the computation in E_a, one value is drawn in the beginning of the run ($\hat{\sigma}_r$) and one in each generation (σ_r). This type of error was used by Akcan et al. [2] in the GDL experiments. Since the measurement of the orientation is considered to be difficult, this rotational error is very interesting.

	variable	Low	High
Range	σ_m	0.010	0.20
	σ_a	0.050	0.30
	$\hat{\sigma}_m$	0.015	0.06
	$\hat{\sigma}_a$	0.150	1.00
	p_{fail}	0.050	0.30
Velocity	σ_m	0.050	0.20
	σ_a	0.050	0.30
	$\hat{\sigma}_m$	0.015	0.06
	$\hat{\sigma}_a$	0.050	0.30
	p_{fail}	0.050	0.20
	σ_r	0.050	0.20
	$\hat{\sigma}_r$	0.050	0.20

Table 4.3: Error Parameters

In the first experiment in subsection 4.5.1 the effect of the error parameters is tested in isolation. In the other experiments three sets of injected errors are used: No, low and high error. The low and high error are chosen so, the algorithm works very well with low error and struggles to find accurate localisation estimates with high error. Each variable in the error levels has been chosen to contribute roughly similar amounts to the overall localisation error, while the multiplicative σ_r ranging error has the most significant impact

of all the error parameters[1]. All parameter values for low and high error are shown in Table 4.3.

4.5 Experiments with Random Walk Motion

The original experiments on GDL presented by Akcan et al. [2] used a random walk motion to test the algorithm. The particles are placed in an area of 100 x 100 units and move with a random speed $|\vec{v}_i(t)| = \mathcal{U}(0, 5)$ in a randomly chosen direction. Those parameters are used for all experiments using the random walk motion. A direct comparison between GDL and the three variants of the localisation algorithm was implemented as well. With the settings proposed by Akcan et al. [2] the SLO localisation performs very poorly. The mean absolute localisation error of the three algorithms was either very close to the communication radius used by the particles, or much higher. This means the localisation does not work at all. A randomly chosen point within the communication radius has a similar amount of localisation error.

There are two reasons why the SLO localisation method does not work with this combination of communication radius and movement speed. First, the communication radius is chosen very small compared to the speed of the particles, so neighbourhood changes are very frequent. As GDL does not use past results in its computation at all, it can handle this. On the other hand SLO uses knowledge about prior generations and therefore needs to converge to the correct solution before the result is usable. Second, GDL does not provide a localisation result when an error amplifying state is detected (up to 16% of values in the experiments) [2]. By filtering results with higher error, GDL can perform much better than SLO in this scenario. SLO provides an estimate even if an error amplifying input was detected. The value used by SLO in the error mitigation case is obtained by dead-reckoning. Because of the frequent neighbourhood changes, the previous position is often unknown and the dead-reckoning is even more disadvantageous.

Trading the disadvantages of the SLO method for the loss of erroneous estimates seems undesirable. However, the missing values are highly problematic in robot control. When the input to the controller is missing, the robot can

[1] This error has been determined to be a key influence in the error experiment, likely to influence the hardware choice for distance sensors.

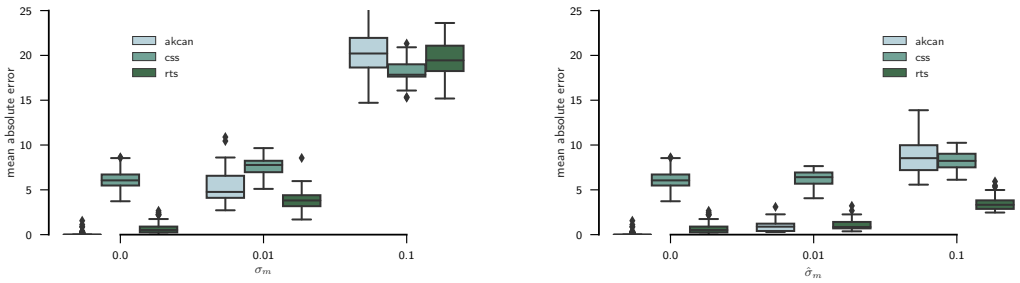
not decide where to move which has several undesirable consequences. The worst problem is that the robot can not even stop when values are missing, because further movement is needed to localise the robot in the future. If the convergence does not stop prematurely, the SLO algorithm may also be able to converge closer to the real value than GDL as more information is exploited in the computation.

While SLO was clearly outperformed in this experiment, the other experiments show that SLO works, but a comparison to GDL [2] is not possible. The factors that are beneficial and harmful to its performance are also analysed in more detail in those experiments.

4.5.1 Error

The first parameters examined are the error parameters shown in Table 4.3. In this experiment all error parameters were set to zero while one by one the parameters were set to different values. In prior experiments with low swarm size and fewer runs with each parameter, the values for each parameter that should be examined were determined. This process has been repeated for all estimation methods and neighbourhoods. All other parameters are not modified.

The most severe error type is the stochastic value correlated error (σ_m) in the range measurements. The same input error value leads to lower localisation errors for all other parameters. The experiment showing the influence of this error is depicted in Figure 4.1. It is very interesting that the permanent error (4.1b) is much less severe than the stochastic error (4.1a). This can be explained by the geometry of the triangle used for the position estimation (see subsection 3.5.1). Generally, the distance between the points are likely to be larger than the velocity, resulting in a triangle with two long sides for d_t and d_{t-1} and a short side for $\vec{v}$. When the d_{t-1} and d_t are both modified by the same value, the triangle is stretched (or shortened), but the angles in the triangle are not changed by a large amount (especially in such a "thin" triangle). When the range error changes between generations, d^*_{t-1} can be stretched, while d^*_t is reduced. In case the error is changing the angles adjacent to the short side of the triangle change much more compared to a none changing error. As a consequence, the direction of $\vec{v}^*$ in the triangle changes more, resulting in a larger localisation error.



(a) Dynamic (b) Constant

Figure 4.1: Multiplicative Range Error. The mean absolute error as it is affected by the dynamic multiplicative input error σ_m (a) and the constant multiplicative input error $\hat{\sigma}_m$ (b).

The estimation error generated by the additive range error is smaller in magnitude than the multiplicative error because in the random walk scenario, the distances are very long. Therefore, in the random walk scenario the effect of the additive errors is negligible (when multiplicative errors are present). In the velocity measurement the additive errors are more relevant. One reason is that the combined velocity vector $\vec{v}^*$ is usually shorter than the distance between the particles. More importantly, the additive velocity error is one of the main error types causing runaway configurations in the RTS selection method. As is described in section 3.5.3 the estimate can deteriorate in a special case: When an estimate has been found, the round trip error e_r is computed for three points. This round trip is used to select the correct localisation estimates. However, one of the estimates is the position updated with the velocity vectors. As this estimation method is a perfect computation, e_r does not change, even if the velocity used in the computation contains errors. This means the dead-reckoning estimate $\vec{x}_3^*$ is always used and the quality of the solution deteriorates.

Figure 4.2 shows the mean absolute error for the experiments executed with the constant, additive velocity error. In the RTS selection method the effect of the runaway configurations can clearly be seen in the fully connected- and ring neighbourhood. As those neighbourhoods are static, the probability of finding a new configuration with lower e_r is very low. In changing neighbourhoods (knn, random, communication radius) the error is less severe. When the neighbourhood changes, another round trip is evaluated – and for this triangle the

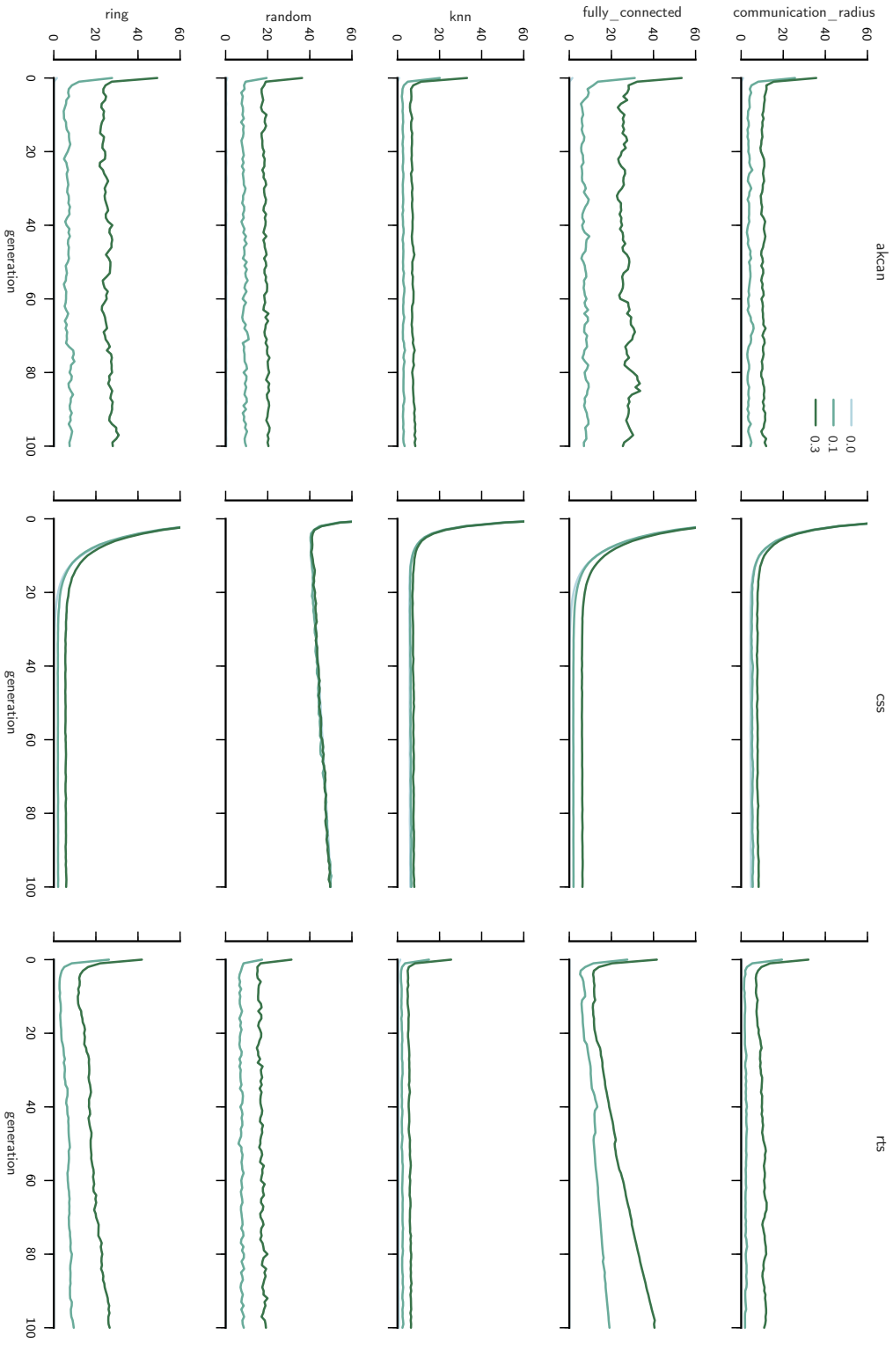
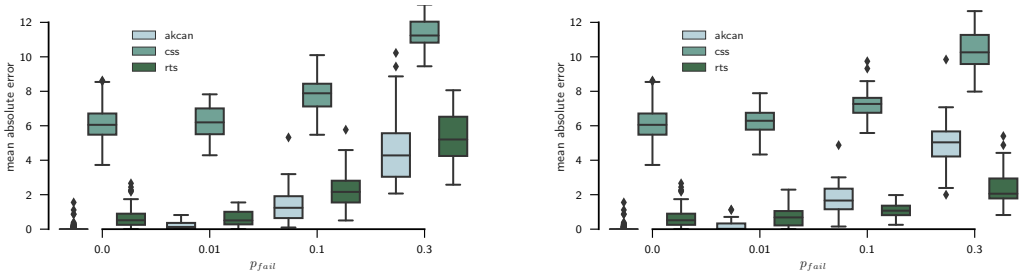


Figure 4.2: Influence of a Fixed Velocity Offset. Mean absolute error for 100 generations for each neighbourhood and selection method.

best localisation (in terms of e_r) is more likely to be one of the good estimates. The impact of the neighbourhood is discussed in more detail in section 4.5.3. However, it is very interesting that the effect of the input errors on localisation accuracy is strongly affected by the neighbourhood relationship. The RTS selection method copes comparatively well with the random neighbourhood – the nemesis of the CSS method. This means that the constant, permanent velocity error is problematic for the RTS selection method in a static neighbourhood. The CSS method is more affected by all circumstances that slow down or reset convergence and this can be caused by errors as well as changes in the neighbourhood. Thus, the selection methods should be combined to compensate for each other's weakness.



(a) Velocity

(b) Range

Figure 4.3: Localisation Error Caused by Sensor Failures

The localisation algorithm is able to deal with sensor failures remarkably well. Simulated sensor failures are controlled by the p_{fail} parameter. The range sensor and the velocity sensor can both fail independently. When a simulated sensor failure occurs, zero is returned as a measurement. Figure 4.3 shows the impact of sensor failures on the localisation error. Even a 30% failure rate results in an acceptably small localisation error. In case of a range sensor failure the new value is computed by dead-reckoning. Therefore, the new value is correct if the last estimate is correct. When the velocity sensor fails, the velocity of one of the nodes is zero. Hence, the combined velocity $\vec{v}^*$ of two nodes is computed incorrectly and this part of the localisation is erroneous. The dead-reckoning uses the old position of the other node as its new estimate. This keeps the algorithm working. It also explains why the CSS method is outperformed in those cases by such a large margin, and why RTS

and Akcan's selection can deal with range sensor failures better than velocity sensor failures.

4.5.2 Mitigation

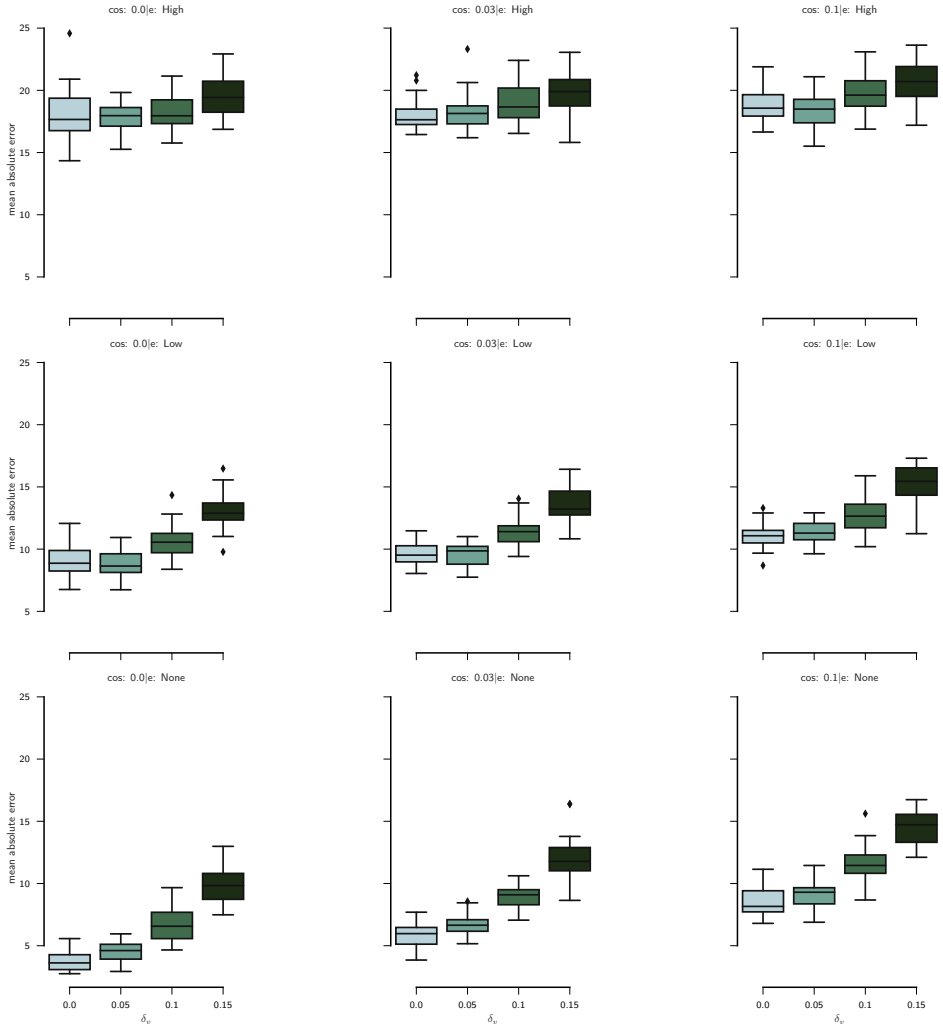
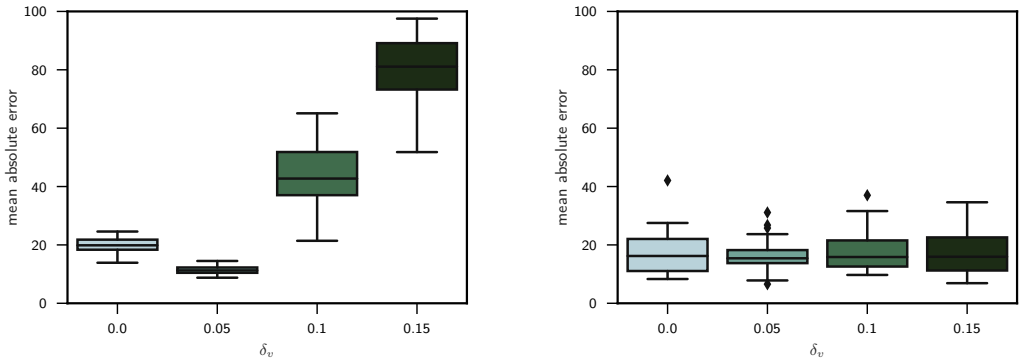


Figure 4.4: Mitigation Effects for CSS with knn-Neighbourhood

The mitigation parameters have been used in an earlier implementation of the algorithm with good results [30]. Nevertheless, it is reasonable to evaluate



(a) CSS (b) RTS

Figure 4.5: Mitigation in ring neighbourhood CSS and RTS, with low error.

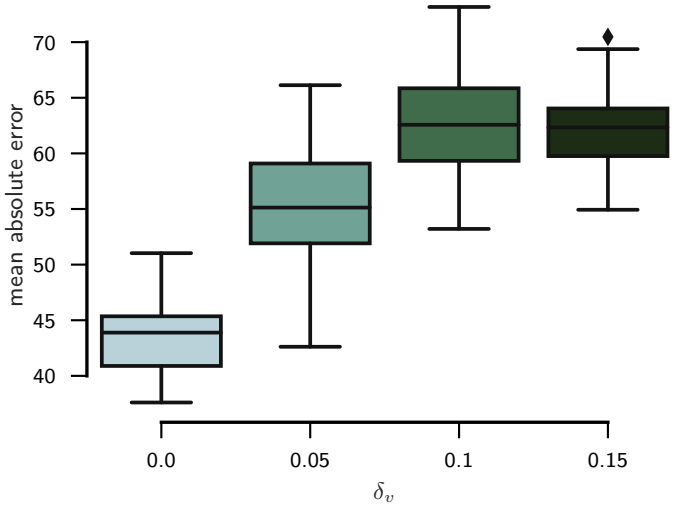


Figure 4.6: Mitigation in Random Neighbourhood, CSS, no error

δ_{cos} and δ_v in the current implementation, especially as only the mitigation with both values together has been tested before. δ_{cos} filters values close to the singularity that occurs when the cosine in the computation can not be computed due to measurement errors ($|cos(\alpha)| > 1$ can not be solved for α). δ_v filters values that are part of an error amplifying configuration that occurs when the particles move in parallel or too slowly. Section 3.5.2 describes how the mitigation parameters work.

The mitigation parameters are tested with all neighbourhoods, the selection methods and three levels of error (None, Low and High). The selection method that is most affected by the mitigation parameters is the CSS selection, hence the focus in this section is on the effectiveness of error mitigation with CSS selection. The CSS method is most affected because the other two selection methods have an internal, error-based metric to filter the same values the mitigation method would filter too. While with the CSS method both the quality of the solution in a converged state and convergence rate are strongly affected by the mitigation parameters. The other selection methods can also benefit from the error mitigation by saving computation power. This means as long as the mitigation is not harmful, it can be considered to be a positive component of the algorithm.

Figure 4.4 shows the mean absolute localisation error at the 100^{th} generation for all tested values of the localisation parameters. The knn-neighbourhood was used. In each row the input error is varied (top: High, bottom: None). As both δ_{cos} and δ_v can have negative effects, the values are chosen small enough to search for an optimal value. The experiment shows that when δ_{cos} is varied, it has almost no effect on the localisation error (as shown in each column of the Figure 4.4). Only in the error-free configuration a clear effect can be observed. When more error mitigation is added in an error-free scenario, a negative impact is not surprising. As the effect is so small, the middle value of 0.03 was chosen as the default value for δ_{cos} in the other experiments. The reason why δ_{cos} is not set to zero is the assumption that no negative impact on localisation accuracy is still positive in terms of computation effort (computational effort was not measured, tough). The more important mitigation parameter is δ_v. Akcan et al. [2] use a similar mitigation method in GDL. In Figure 4.4 on the x-axis of each diagram δ_v is varied. When no input error is present (bottom row), the negative impact of the mitigation is evident. However, the plots with low and high input error (middle and top row) show that a small value for δ_v sometimes reduces the output error. When higher levels for δ_v are used, the slower convergence of the algorithm combined with the errors in the dead-reckoning outweigh the benefits of the error mitigation again.

Error mitigation has two drawbacks: The error mitigation should filter the output when the configuration is known to amplify input errors. Yet, a high input error can still produce wrong output in unfiltered cases. Similarly, small errors that are amplified might still lead to tolerable output errors. While the experiments show that the overall error can still be reduced by filtering those

configurations, convergence speed of the method is lowered by filtering values. This is especially a problem in the CSS method.

In Figure 4.4 the knn-neighbourhood was used – a dynamic neighbourhood that can change in between generations. Figure 4.5a shows the mean localisation error of the algorithm with the ring topology (static) and low error. Here the mitigation is more useful, because lower convergence rate is a smaller problem in the static neighbourhood. In contrast to that, Figure 4.6 shows that the performance in the random topology is diminished completely by the mitigation. The impact of events that reset convergence is observable in all of the experiments. Those events are an important factor to the algorithm's performance.

4.5.3 Neighbourhood

The experiments testing the behaviour of the algorithm in different neighbourhoods and different parameters confirm the findings of the previous experiments. Especially the static ring neighbourhood and the random topology are interesting to examine as they are designed to show the effects of connectedness and neighbourhood changes. At the same time ring and random neighbourhood are not prone to the bias that knn and communication radius introduce by filtering for particle distance. The ring neighbourhood is interesting as it is a static neighbourhood (no nodes enter/leave the neighbourhoods) and the number of connections in the graph can be modified by the parameter n. In contrast to that the random topology is very dynamic. In the random topology the parameter p represents the connection probability that is re-evaluated for each pair of particles in each generation. Higher values of p mean less change in the neighbourhood of each node as the probability for a node to vanish from the neighbourhood is $1 - p$.

In Figure 4.7 the influence of the connectedness parameter n on the absolute localisation error is shown for the CSS selection method. The values clearly show that the connectedness has close to no influence on localisation accuracy. The influence on the other variants of the algorithm is similarly insignificant. Figure 4.8a shows similar algorithm runs with three different p values for the random topology for CSS and RTS selection. The localisation error for different

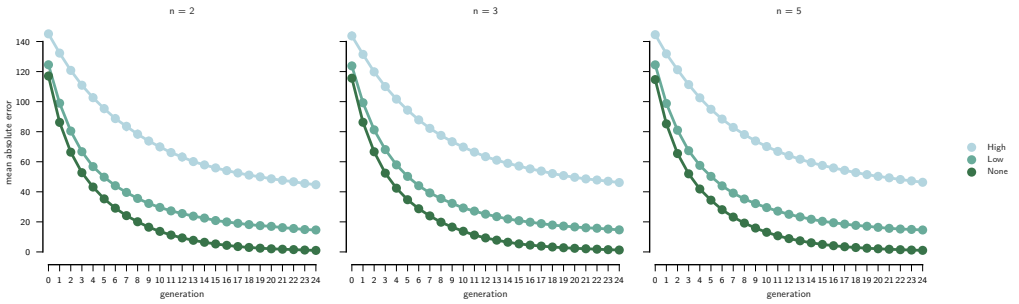
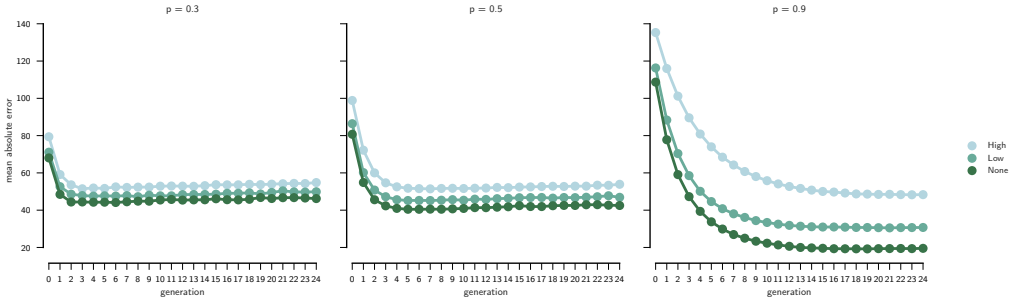
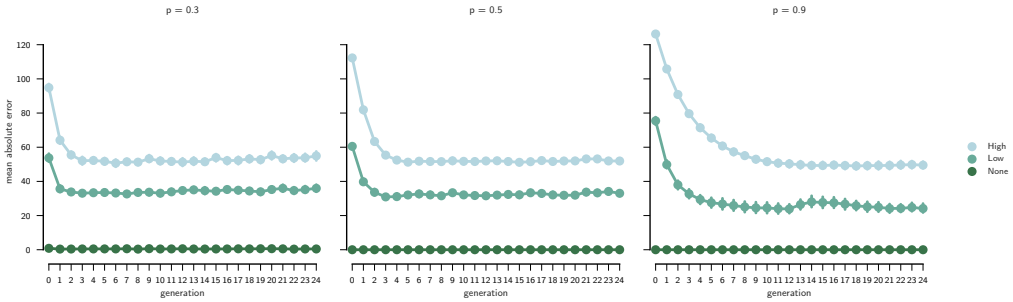


Figure 4.7: Ring Neighbourhood with CSS selection



(a) CSS



(b) RTS

Figure 4.8: Random Neighbourhood with CSS selection

connection probabilities p in the random topology is shown in Figure 4.8. The performance of the algorithm with the CSS method (Figure 4.8a) is clearly affected by p. Not only the overall performance is interesting, but also the convergence behaviour of the algorithm. While the error after 25 generations is much smaller with $p = 0.9$, the error is smaller during the first five generations for $p = 0.3$ and $p = 0.5$. This indicates that the strategy for assigning a position to particles entering the neighbourhood produces a

smaller error than the random initialisation in the beginning of the algorithm. The same effect on convergence behaviour can be observed in the RTS method Figure 4.8b. Apart from that, the algorithm seems largely unaffected by the random neighbourhood changes when RTS is used. The RTS method clearly outperforms CSS in the random topology. This is explained by the fact that particles added to the neighbourhood benefit from the localisation information in the rest of the swarm that is not used in the CSS method.

4.5.4 Prediction Weight

As the CSS selection method contains a parameter, this parameter should be examined in an experiment. The parameter in question is the weight w that decides which solution to chose in a weighted average. CSS selection method is explained in more detail in section 3.5.3.

w should be chosen from the range $[0, 1]$. The higher w is, the more influence is given to the dead-reckoning with the velocity vector. This means high values of w are best used when the prior estimates are known to be good. The estimates are especially poor when new neighbours are added to the neighbourhood. Therefore, a rapidly changing neighbourhood such as the random topology calls for lower values of w compared to static topologies such as the fully connected or ring neighbourhood. A factor that should also influence the value of w is the quality of the distance sensor. More accurate distance sensors allow smaller values of w. An accurate measurement of the distance only improves the geometric solutions, not the dead-reckoning.

A value of 0.25 to 0.5 seems to be most reasonable. 0.5 was chosen as the value for the other experiments using the CSS method as it seems to be a good trade-off between convergence speed and accuracy enhancement. In Figure 4.10 the localisation error is plotted over time for different values of w. While the localisation converges slower, a higher value for w results in more accurate localisation as more time passes. The intervals where different values w work best are most interesting. $w = 0$ works best for the first ten generations. From ten to 20 generations 0.25 is the best value. After the 50^{th} generation $w = 0.75$ is the best value. This also means that w is tuned for neighbourhoods that change in average every 20 to 50 generations. This puts the chosen trade-off more towards static neighbourhood configurations and partly explains low accuracy in changing neighbourhoods.

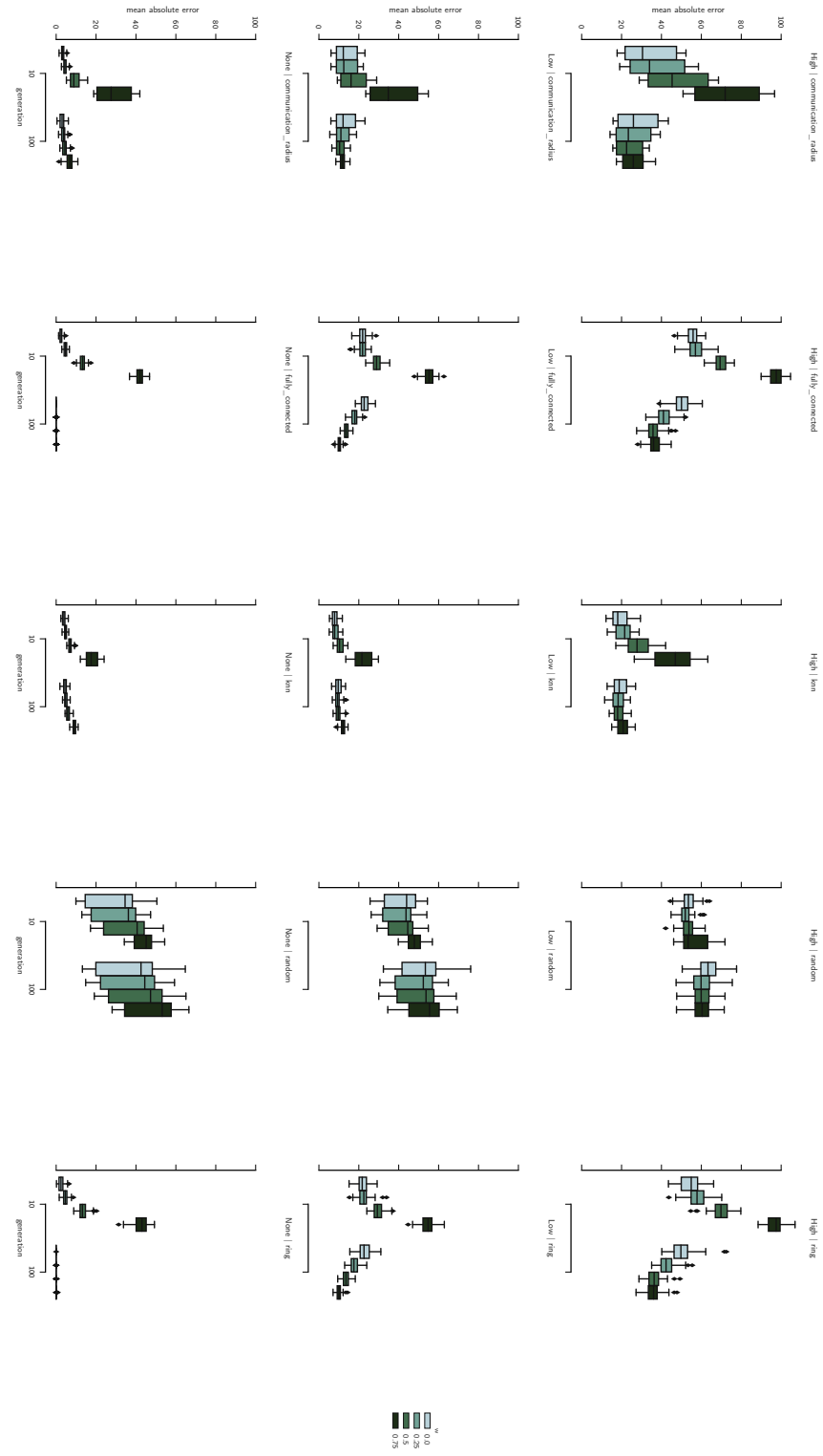


Figure 4.9: Estimation Error for Different Values of the Prediction Weight

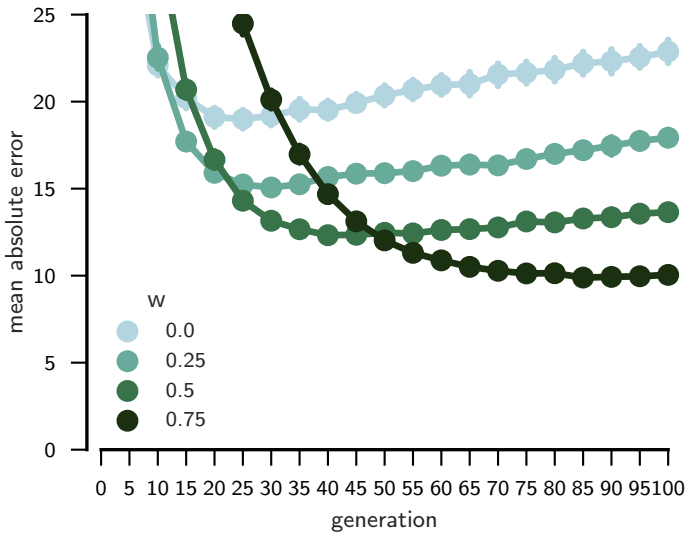


Figure 4.10: Estimation Error for Different Values of the Prediction Weight in Ring Neighbourhood

4.6 Experiments with PSO Motion

The last experiment was run with the full algorithm, containing the localisation algorithm as well as the PSO. This experiment is the most complex, this means interpreting the results is more sophisticated than in the prior experiments. Nevertheless, the experiment is important as it shows that the components of the algorithm work well together. The most interesting question that should be answered by the experiment's results is whether the location estimates deteriorate the performance of the PSO algorithm. Additionally, the movements generated by the PSO algorithm affect the movement-based localisation algorithm. Therefore, it is important to show that the localisation works with the movement vectors generated by the PSO algorithm.

Multiple parameters are specific to the PSO movement that was used in this experiment. First, the parameters to the PSO algorithm must be determined. Since the goal of this thesis is not to find the best PSO parameters, the default parameters suggested in [56] have been used for those parameters. The additional parameters $c_3 = 1$ and $c_4 = 0$ have been chosen, so they are mathematically neutral (multiply by 1 and add 0). Furthermore, the PSO must find

a minimum in a fitness landscape. Therefore, an objective function is needed to test the algorithm. The following standardised benchmark functions were used: Ackley, Griewank, Line, Plane, Rastrigin, Rosenbrock and Sphere. A short summary for each of the functions is given in the documentation of the deap-python package [7] that was used as an implementation. All objective functions are minimisation functions.

Table 4.4 shows how the PSO algorithm performs with the knn-neighbourhood. The fact that the algorithm often performs better when errors are introduced into the range and velocity measurements is most interesting. This fact is explained by conceptualising the performance as a combination of exploration and exploitation behaviour. Exploration is the algorithm's ability to cover a large area and find any optimum beyond the previously known landscape. Exploitation is the behaviour that lets the algorithm converge towards an optimum in a known area.

The sphere function is an objective that is focused on exploitation, as a gradient is present in all of the search space and there are no local optima. Additionally, the gradient points directly to the optimal solution. The Ackley function on the other hand has many local optima that can only be overcome by exploration. The results of the PSO runs in Table 4.4 clearly show that the algorithms perform better in the exploration objectives and worse in the exploitation objective if the range and velocity measurements are infused with an error. This is also reflected in the fitness values at the end of the run, during the run and in the number of generations needed to reach a fitness threshold predefined for each objectives. In the Sphere objective the error-less run yields a much better result than the runs with error and there is a clear difference between low and high error. In the Ackley objective the opposite is the case, as the error-free variant of the algorithm does not explore the search space enough to find the optimal solution.

Another effect observable is that there is a lower bound to the fitness the simulation is able to reach in case an error is present. This is explained by the additive error component E_a that is the dominant source of error when the particles are very close to each other. The median distance between the particles is shown in Figure 4.11. In Figure 4.12 the median relative error for all estimates is shown for the same experiment runs. The error drops in the first few generations, the particle distance decreases. After about 55 generations, the error starts to increase. At the same time, the particles are

Table 4.4: PSO performance with knn neighbourhood. Generations to reach quality criteria, minimal fitness per run, minimal fitness of the last generation. All values: The median for the runs with the given parameters.

		None			Low			High		
		gen	fitness	fitness end	gen	fitness	fitness end	gen	fitness	fitness end
ackley	akcan	inf	7.035e+00	7.035e+00	117.5	9.570e-03	1.068-02	269.5	2.102e-02	2.342e-02
	css	46.0	5.154e-03	5.154e-03	76.0	6.798e-03	7.659-03	402.5	4.785e-02	4.785e-02
	rts	inf	3.186e+00	3.186e+00	164.5	1.017e-02	1.244-02	478.0	3.312e-02	4.804e-02
griewank	akcan	inf	1.507e-01	1.775e-01	inf	5.385e-02	6.656-02	514.5	5.729e-03	7.514e-03
	css	inf	8.360e-02	1.060e-01	inf	6.350e-02	7.121-02	414.0	5.802e-03	7.729e-03
	rts	inf	6.961e-02	1.060e-01	inf	6.294e-02	8.630-02	638.5	5.293e-03	7.439e-03
line	akcan	30.5	8.592e-08	1.129e-05	69.5	9.949e-05	1.127-01	194.5	2.922e-04	4.852e-01
	css	40.5	9.555e-09	4.565e-07	54.5	2.405e-04	2.148-01	150.5	7.472e-04	5.558e-01
	rts	31.0	2.407e-07	4.884e-05	67.0	1.154e-04	5.930-02	144.5	8.424e-04	1.149e+00
plane	akcan	inf	-6.420e+01	-6.420e+01	inf	-3.086e+02	-3.086+02	179.5	-3.575e+07	-3.575e+07
	css	inf	-9.4378+01	-9.437e+01	inf	-3.582e+02	-3.582+02	171.0	-1.094e+05	-1.094e+05
	rts	inf	-8.2907+01	-8.290e+01	inf	-2.508e+02	-2.508+02	191.5	-2.446e+04	-2.446e+04
rastrigin	akcan	inf	5.097619e-01	5.475e-01	105.5	2.205e-04	3.372-04	134.5	4.363e-05	4.363e-05
	css	181.5	4.617e-05	4.617e-05	109.5	6.441e-04	8.947-04	172.5	3.947e-03	4.530e-03
	rts	inf	9.949e-01	9.949e-01	172.0	5.831e-04	5.831-04	141.5	2.617e-05	3.043e-05
rosenbrock	akcan	12.0	2.067e+01	2.275e+01	11.0	4.906e+00	8.875+00	16.0	4.739e-03	1.007e-02
	css	12.0	5.891e+00	9.692e+00	15.0	2.798e+00	3.053+00	16.0	6.175e-03	8.056e-03
	rts	11.0	1.386e+01	1.809e+01	11.0	3.177e+00	6.709+00	13.0	5.356e-03	8.284e-03
sphere	akcan	19.0	1.005e-151	1.005e-151	26.5	7.096e-10	7.096-10	56.5	5.257e-10	5.257e-10
	css	25.5	6.484e-140	6.484e-140	30.0	8.178e-08	3.493-07	59.0	3.192e-06	5.885e-06
	rts	21.0	7.598e-174	7.598e-174	22.0	2.835e-10	4.323-10	55.5	3.026e-10	3.026e-10

still moving closer to each other, but at a much smaller rate than before. As the particles are moving closer to each other, the additive error component becomes dominant and localisation accuracy diminishes. The fitness will still improve, even after the localisation degenerated. However, most improvement is observed in the first generations where the localisation still works well, as can be seen in Figure 4.13. The fitness improves even after the localisation deteriorates because of the cognitive component. The previous best solution improves, while the particles are moving pseudo-randomly. Therefore, the particles are still able to move closer to each other and improve their fitness at a slower rate.

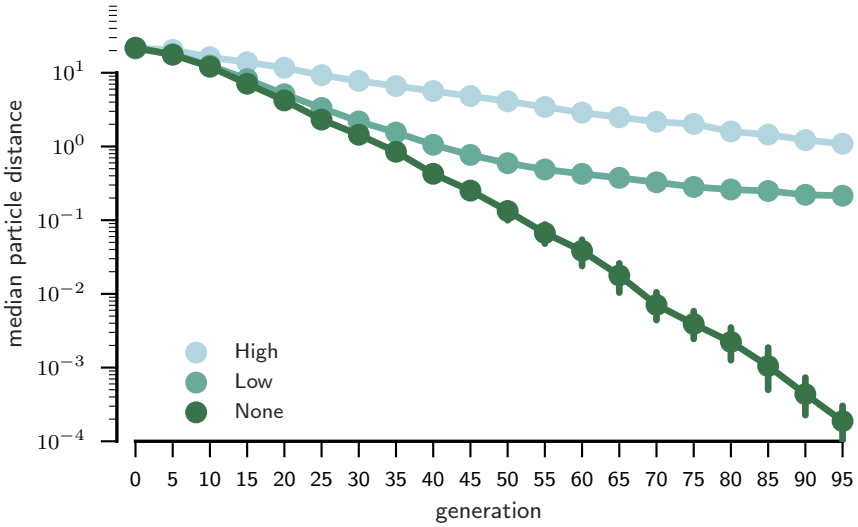


Figure 4.11: Median distance of the particles in the sphere objective with knn neighbourhood and CSS.

The minimal precision of the PSO algorithm that is caused by the relation between particle distance and localisation accuracy is one of the most important findings of this thesis and should be subject of future research. On one hand, the minimal precision may not cause problems, as robotic applications must involve mechanisms for collision avoidance and the distance between the robots is strictly limited by the size of the robots, even if collisions are allowed within the robots' specification. On the other hand, minimal precision may be a critical variable in the design of an application. Especially when the localisation is used to enhance the accuracy of other localisation systems such as

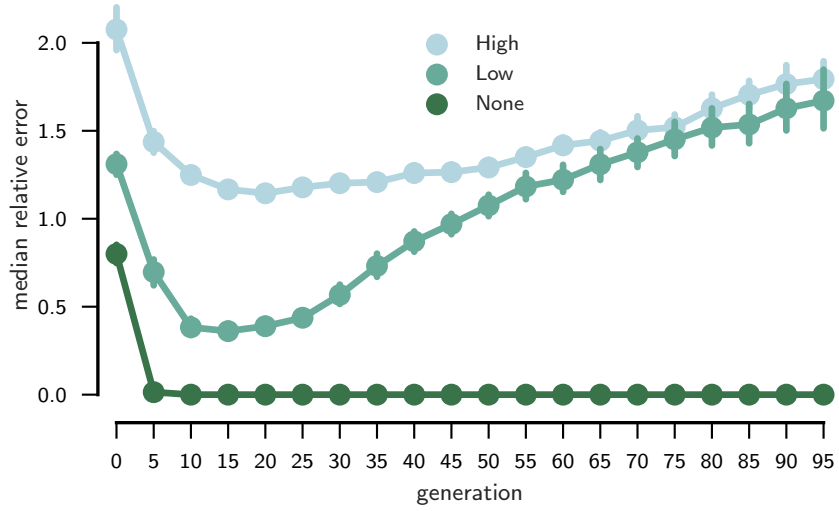


Figure 4.12: Relative localisation error for the same run as Figure 4.11.

GPS, minimal precision may be a key factor in the decision for or against a localisation method.

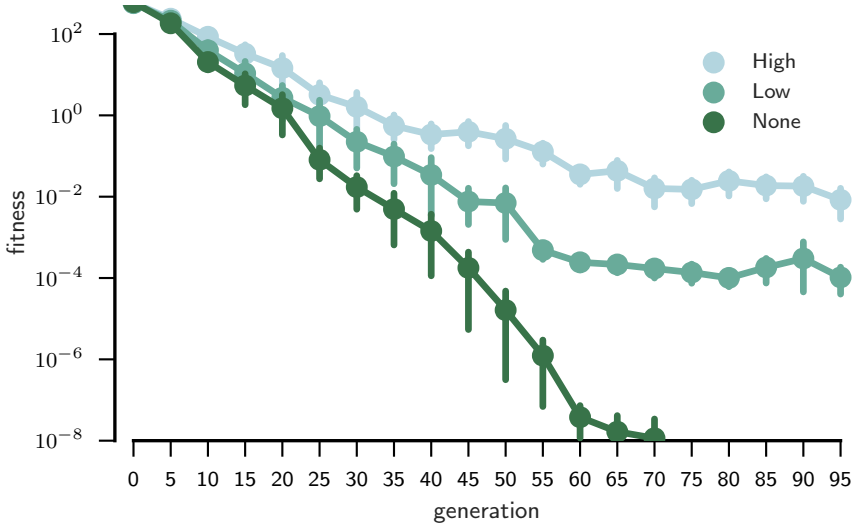


Figure 4.13: Minimal fitness for each generation, using the same runs as Figure 4.11

4.7 Discussion of Results

The results of the experiments are very promising. The algorithm works: The individual components have been implemented and thoroughly tested and errors can be handled reasonably well. Unfortunately, the performance of the algorithm can not be compared to the GDL [2] algorithm, as the scenarios used in the GDL experiments are not within the parameters where SLO works well. However, the algorithm is well suited for an application if the necessary sensors are present and no simple localisation scheme like GPS can be used. Additionally, the algorithm can act as a blueprint for other implementations working with different swarm intelligence algorithms.

The most important findings of the evaluation are twofold: Ranging errors affect the algorithm most severely, especially multiplicative noise is a main source of localisation error. Moreover, permanent errors have a smaller impact on localisation accuracy than stochastic errors. Sensor failures can be handled very well by the algorithm.

The PSO experiments have revealed that the localisation deteriorates when the particles move to close to each other. This is especially important in the design of new robotic systems, when choosing components and specifying the

operating environment. However, this property of the algorithm is only a small disadvantage. In a real application where the robots have a fixed size they can not get infinitely close anyway. While the SLO algorithm performs worse in exploitation tasks if an error is present, the exploration behaviour works better in the same cases. This also means that the SLO algorithm automatically starts to explore once the swarm has converged to a single point which may actually be an interesting property in some use-cases.

5 Conclusion

In the thesis a method was created, implemented and tested to embed swarm intelligence algorithms in a sensor-actor-based environment. The first goal was to show that the algorithm works when localisation and PSO are used together. As the experiments show, the algorithms can work together and the implemented algorithm is useful as a blueprint to future implementations.

A second goal was to find out how measurement errors affect the algorithm's results. Thorough experiments have been conducted to analyse this interaction. The most sensitive input is the distance sensor. Errors with a fixed offset have proved to be less severe than errors that randomly change over time.

The third goal was to learn about the general behaviour of the algorithm. Results of this research are the interactions between neighbourhood changes, algorithm parameters and errors that change the convergence rate of the algorithm and the lower boundary for precision.

The relationship between localisation accuracy and particle distance is an important aspect in designing new robotic systems using the SLO method. Therefore, more research should be conducted to measure whether the method could be applied in a scenario with given accuracy requirements. A new criterion must be defined to easily compute the accuracy with given input parameters.

While the concept has worked in a simulation, an implementation with real robots, or at least a more realistic simulation with a simulation framework like v-rep or similar tools is needed to find out about more pitfalls of real world implementations of the algorithm. In such an implementation collision avoidance is a major component that must be added to the algorithm. Collision avoidance by means of attraction repulsion function as described by Gazi [12] is a topic that could also be examined on the same level of abstraction and the same simulation used in this thesis.

There are also advanced concepts that may improve other parts of the algorithm. The CSS method works and is easy to implement, but a particle filter

tracking multiple generations or a Kalman filter should be able to outperform the CSS selection method, while maintaining the low communication footprint. Introducing asynchronous sensor readings may complicate RTS and Akcan's selection method. As the computation of the error metrics used relies on many individual sensor readings that can only be aggregated over a span of time, the first measurements are outdated as the computation takes places. This problem could be mitigated by running a combination of the selection methods. CCS could be updated with a relatively high frequency, while RTS or Akcan's selection method can be used to find good initial configurations or rectify dead-reckoning errors.

A more general research topic that has come up during the research of this paper lies in the performance metric of the swarm intelligence algorithms. Comparisons between swarm intelligence algorithms usually assume that function evaluations determine the cost of gathering information. This assumption is reasonable when solving numerical problems. However, reading a sensor in a robot is usually a cheap and quick operation, while movement is costly. This means a new benchmark criterion for swarm algorithms based on the distance the particles travel may be very beneficial in deciding for a PSO algorithm in a robotic application [42].

The SLO method was created to apply swarm intelligence algorithms in robot control by satisfying the information need of swarm intelligence algorithms. Hopefully, the method is a significant step towards swarm robotics.

Bibliography

[1] Hüseyin Akcan and Cem Evrendilek. Gps-free directional localization via dual wireless radios. *Computer Communications*, 35(9):1151 – 1163, 2012. Special Issue: Wireless Sensor and Robot Networks: Algorithms and Experiments.

[2] Hüseyin Akcan, Vassil Kriakov, Hervé Brönnimann, and Alex Delis. GPS-Free node localization in mobile wireless sensor networks. In *Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access*, pages 35–42. ACM, 2006.

[3] Hüseyin Akcan, Vassil Kriakov, Hervé Brönnimann, and Alex Delis. Managing cohort movement of mobile sensors via gps-free and compass-free node localization. *Journal of Parallel and Distributed Computing*, 70(7):743 – 757, 2010.

[4] Tim M. Blackwell and Peter J. Bentley. Dynamic search with charged swarms. In *GECCO*, 2002.

[5] Srdjan Čapkun, Maher Hamdi, and Jean-Pierre Hubaux. Gps-free positioning in mobile ad hoc networks. *Cluster Computing*, 5(2):157–167, Apr 2002.

[6] Kok Seng Chong and Lindsay Kleeman. Accurate odometry and error modelling for a mobile robot. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 4, pages 2783–2788. IEEE, 1997.

[7] François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau, and Christian Gagné. Deap: A python framework for evolutionary algorithms. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, pages 85–92, New York, NY, USA, 2012. ACM.

[8] S. Doctor, G. Venayagamoorthy, and V. Gudise. Optimal pso for collective robotic search applications. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 1390–1395, 2004.

[9] Marco Dorigo and Thomas Stützle. Ant colony optimization: Overview and recent advances. Technical Report TR/IRIDIA/2009-013, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, May 2009.

[10] Andries P. Engelbrecht. Heterogeneous particle swarm optimization. In Marco Dorigo, Mauro Birattari, Gianni A. Di Caro, René Doursat, Andries P. Engelbrecht, Dario Floreano, Luca Maria Gambardella, Roderich Groß, Erol Şahin, Hiroki Sayama, and Thomas Stützle, editors, *Swarm Intelligence*, pages 191–202, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[11] Yasutaka Fuke and Eric Krotkov. Dead reckoning for a lunar rover on uneven terrain. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 1, pages 411–416. IEEE, 1996.

[12] Veysel Gazi and Kevin M. Passino. A class of attractions/repulsion functions for stable swarm aggregations. *International Journal of Control*, 77(18):1567–1579, 2004.

[13] Veysel Gazi and Kevin M. Passino. *Swarm stability and optimization*. Springer, Berlin, 2011. OCLC: 711872796.

[14] J. Hightower and G. Borriello. Location systems for ubiquitous computing. *Computer*, 34(8):57–66, Aug 2001.

[15] Ian A.R. Hulbert and John French. The accuracy of GPS for wildlife telemetry and habitat mapping. *Journal of Applied Ecology*, 38(4):869–878, August 2001.

[16] Rajagopal Iyengar and Biplab Sikdar. Scalable and distributed GPS free positioning for sensor networks. In *Communications, 2003. ICC'03. IEEE International Conference on*, volume 1, pages 338–342. IEEE, 2003.

[17] W Jatmiko, F Jovan, RYS Dhiemas, Alvissalim M Sakti, Fanany M Ivan, T Fukuda, and K Sekiyama. Robots implementation for odor source localization using pso algorithm. *WSEAS Transactions on Circuits and Systems*, 10(4):115–125, 2011.

[18] Wisnu Jatmiko, Petrus Mursanto, Benyamin Kusumoputro, Kosuke Sekiyama, and Toshio Fukuda. Modified pso algorithm based on flow of wind for odor source localization problems in dynamic environments. *WSEAS Transaction on System*, 7(3):106–113, 2008.

[19] L. Jayatilleke and N. Zhang. Landmark-based localization for unmanned aerial vehicles. In *2013 IEEE International Systems Conference (SysCon)*, pages 448–451, April 2013.

[20] Yi Jiang and Victor CM Leung. An asymmetric double sided two-way ranging for crystal offset. In *Signals, Systems and Electronics, 2007. ISSSE'07. International Symposium on*, pages 525–528. IEEE, 2007.

[21] Wei-Wen Kao. Integration of gps and dead-reckoning navigation systems. In *Vehicle Navigation and Information Systems Conference, 1991*, volume 2, pages 635–643, Oct 1991.

[22] Benjamin Kempke, Pat Pannuto, and Prabal Dutta. Polypoint: Guiding indoor quadrotors with ultra-wideband localization. In *Proceedings of the 2Nd International Workshop on Hot Topics in Wireless*, HotWireless '15, pages 16–20, New York, NY, USA, 2015. ACM.

[23] Lindsay Kleeman. Optimal estimation of position and heading for mobile robots using ultrasonic beacons and dead-reckoning. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 2582–2587. IEEE, 1992.

[24] Tomáš Krajník, Matías Nitsche, Jan Faigl, Petr Vaněk, Martin Saska, Libor Přeučil, Tom Duckett, and Marta Mejail. A practical multirobot localization system. *Journal of Intelligent & Robotic Systems*, 76(3):539–562, Dec 2014.

[25] Thiemo Krink, Bogdan Filipic, Gary B Fogel, and René Thomsen. Noisy optimization problems-a particular challenge for differential evolution? In *IEEE congress on Evolutionary Computation*, pages 332–339. Citeseer, 2004.

[26] K. N. Krishnanand and D. Ghose. *A Glowworm Swarm Optimization Based Multi-robot System for Signal Source Localization*, pages 49–68. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[27] Dikai Liu, Lingfeng Wang, and Kay Chen Tan, editors. *Design and Control of Intelligent Robotic Systems*, volume 177 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[28] Sergei Lupashin, Markus Hehn, Mark W Mueller, Angela P Schoellig, Michael Sherback, and Raffaello D'Andrea. A platform for aerial robotics research and demonstration: The flying machine arena. *Mechatronics*, 24(1):41–54, 2014.

[29] Y. Ma and E. C. Kan. Accurate indoor ranging by broadband harmonic generation in passive nltl backscatter tags. *IEEE Transactions on Microwave Theory and Techniques*, 62(5):1249–1261, May 2014.

[30] Sebastian Mai, Christoph Steup, and Sanaz Mostaghim. Movement-based localisation for pso-inspired search behaviour of robotic swarms. In *Accepted at ANTS 2018 Proceedings*. Springer, 2018.

[31] Sabrina Merkel, Sanaz Mostaghim, and Hartmut Schmeck. Distributed geometric distance estimation in ad hoc networks. In *International Conference on Ad-Hoc Networks and Wireless*, pages 28–41. Springer, 2012.

[32] Sabrina Merkel, Sanaz Mostaghim, and Hartmut Schmeck. Hop count based distance estimation in mobile ad hoc networks – challenges and consequences. *Ad Hoc Networks*, 15:39 – 52, 2014. Smart solutions for mobility supported distributed and embedded systems.

[33] Alan G Millard, James A Hilder, Jon Timmis, and Alan FT Winfield. A low-cost real-time tracking infrastructure for ground-based robot swarms. In *Swarm Intelligence: 9th International Conference, ANTS 2014, Brussels, Belgium, September 10-12, 2014. Proceedings*, volume 8667, page 278. Springer, 2014.

[34] David Moore, John Leonard, Daniela Rus, and Seth Teller. Robust distributed network localization with noisy range measurements. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 50–61. ACM, 2004.

[35] Sanaz Mostaghim, Christoph Steup, and Fabian Witt. Energy aware particle swarm optimization as search mechanism for aerial micro-robots. In *IEEE Swarm Intelligence Symposium, IEEE SSCI 2016*, 2016.

[36] Alireza Nafarieh and Jacek Ilow. A testbed for localizing wireless lan devices using received signal strength. In *Communication Networks and*

Services Research Conference, 2008. CNSR 2008. 6th Annual, pages 481–487. IEEE, 2008.

[37] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–I. Ieee, 2004.

[38] Evangelos Papadopoulos and Michael Misailidis. On differential drive robot odometry with application to path planning. In *Control Conference (ECC), 2007 European*, pages 5492–5499. IEEE, 2007.

[39] Konstantinos Parsopoulos and Michael Vrahatis. Particle swarm optimizer in noisy and continuously changing environments. pages 289–294, 01 2001.

[40] Max Pfingsthorn, Bayu Slamet, and Arnoud Visser. A scalable hybrid multi-robot slam method for highly detailed maps. In *Robot Soccer World Cup*, pages 457–464. Springer, 2007.

[41] Riccardo Poli. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, 2008, 2008.

[42] Jim Pugh and Alcherio Martinoli. Inspiring and modeling multi-robot search with particle swarm optimization. In *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 332–339. IEEE, 2007.

[43] Jim Pugh, Alcherio Martinoli, and Yizhen Zhang. Particle swarm optimization for unsupervised robotic learning. In *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 92–99. IEEE, 2005.

[44] Ahmad Rezaee Jordehi. Particle swarm optimisation for dynamic optimisation problems: a review. *Neural Computing and Applications*, 25(7):1507–1516, Dec 2014.

[45] Dian Palupi Rini, Siti Mariyam Shamsuddin, and Siti Sophiyati Yuhaniz. Particle swarm optimization: technique, system and challenges. *International journal of computer applications*, 14(1):19–26, 2011.

[46] Michael Rubenstein, Christian Ahler, Nick Hoff, Adrian Cabrera, and Radhika Nagpal. Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, 62(7):966 – 975, 2014. Reconfigurable Modular Robotics.

[47] A. R. Jiménez Ruiz and F. Seco Granja. Comparing ubisense, bespoon, and decawave uwb location systems: Indoor performance analysis. *IEEE Transactions on Instrumentation and Measurement*, 66(8):2106–2117, Aug 2017.

[48] Angel Santamaria-Navarro, Joan Sola, and Juan Andrade-Cetto. High-frequency mav state estimation using low-cost inertial and optical flow measurement units. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 1864–1871. IEEE, 2015.

[49] Yannic Schröder, Georg von Zengen, and Lars Wolf. Poster: Nlos-aware localization based on phase shift measurements. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, pages 224–226, New York, NY, USA, 2015. ACM.

[50] H. Song, V. Shin, and M. Jeon. Mobile node localization using fusion prediction-based interacting multiple model in cricket sensor network. *IEEE Transactions on Industrial Electronics*, 59(11):4349–4359, Nov 2012.

[51] M Srinivasan, Shaowu Zhang, M Lehrer, and T Collett. Honeybee navigation en route to the goal: visual flight control and odometry. *Journal of Experimental Biology*, 199(1):237–244, 1996.

[52] Ashitey Trebi-Ollennu, Terry Huntsberger, Yang Cheng, Eric T Baumgartner, Brett Kennedy, and Paul Schenker. Design and analysis of a sun sensor for planetary rover absolute heading detection. *IEEE Transactions on Robotics and Automation*, 17(6):939–947, 2001.

[53] Lei Wang and Qingzheng Xu. GPS-Free Localization Algorithm for Wireless Sensor Networks. *Sensors*, 10(6):5899–5926, June 2010.

[54] Oliver J Woodman and Robert K Harle. Concurrent scheduling in the active bat location system. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 431–437. IEEE, 2010.

[55] Jiuqiang Xu, Wei Liu, Fenggao Lang, Yuanyuan Zhang, and Chenglong Wang. Distance Measurement Model Based on RSSI in WSN. *Wireless Sensor Network*, 02(08):606–611, 2010.

[56] M. Zambrano-Bigiarini, M. Clerc, and R. Rojas. Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements.

In *2013 IEEE Congress on Evolutionary Computation*, pages 2337–2344, June 2013.

[57] J. Zou, S. Gundry, J. Kusyk, C. S. Sahin, and M. Ü. Uyar. Bio-inspired topology control mechanism for autonomous underwater vehicles used in maritime surveillance. In *2013 IEEE International Conference on Technologies for Homeland Security (HST)*, pages 201–206, Nov 2013.

[58] Sebastian Zug. *Architektur für verteilte, fehlertolerante Sensor-Aktor-Systeme*. PhD thesis, Otto-von-Guericke Universität, Magdeburg, 2011.

Declaration of Authorship

I hereby declare that this thesis was created by me and me alone using only the stated sources and tools.

Sebastian Mai

Magdeburg, July 30, 2018