# Sensor Path Planning for Emitter Localization

*Folker Hoffmann*

A dissertation submitted in partial fulfillment
of the requirements for the degree of
**Doctor of Philosophy**
of
**University College London**.

Department of Electronic and Electrical Engineering
University College London

First supervisor: Prof. Hugh Griffiths
Second supervisor: Dr. Matthew Ritchie

February 16, 2023

I, Folker Hoffmann, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

# Abstract

The localization of a radio frequency (RF) emitter is relevant in many military and civilian applications. The recent decade has seen a rapid progress in the development of small and mobile unmanned aerial vehicles (UAVs), which offer a way to perform emitter localization autonomously. The path a UAV travels influences the localization significantly, making path planning an important part of a mobile emitter localization system.

The topic of this thesis is path planning for a UAV that uses bearing measurements to localize a stationary emitter. Using a directional antenna, the direction towards the target can be determined by the UAV rotating around its own vertical axis. During this rotation the UAV is required to remain at the same position, which induces a trade-off between movement and measurement that influences the optimal trajectories.

This thesis derives a novel path planning algorithm for localizing an emitter with a UAV. It improves the current state of the art by providing a localization with defined accuracy in a shorter amount of time compared to other algorithms in simulations. The algorithm uses the policy rollout principle to perform a nonmyopic planning and to incorporate the uncertainty of the estimation process into its decision. The concept of an action selection algorithm for policy rollout is introduced, which allows the use of existing optimization algorithms to effectively search the action space. Multiple action selection algorithms are compared to optimize the speed of the path planning algorithm. Similarly, to reduce computational demand, an adaptive grid-based localizer has been developed.

To evaluate the algorithm an experimental system has been built and the algorithm was tested on this system. Based on initial experiments, the path planning algorithm has been modified, including a minimal distance to the emitter and an outlier detection step. The resulting algorithm shows promising results in experimental flights.

# Acknowledgements

I would like to thank Hugh Griffiths and Matthew Ritchie for their always helpful guidance during the creation of this thesis. I am grateful to Alexander Charlish for his support in all the years I was working in his group. I also would like to thank Wolfgang Koch, whose lecture about sensor data fusion and enthusiasm about the topic motivated me to pursue this field of study.

Building the experimental system, described in Chapter 6, was a joint work between Hans Schily, Markus Krestel and me to which I contributed the path planning algorithm and analysis presented in this thesis. Without them, this experimental system would never have existed. Thank you for the joint work, it was always a pleasure to work together with you! Hans Schily built the emitter and implemented several important ROS components. Markus Krestel implemented the direction finding algorithm and its connection to the ROS system. In the design of the payload, field tests, and all the other work which emerges when building such a system we were all involved. Hans Schily also implemented the algorithm [Vander Hook et al., 2015], which was used as a reference algorithm in Chapter 4.

During the experimental trials in this thesis, I also received valuable help by many other people. With the hope of not omitting anyone, I would like to thank Janning Nettekoven for assembling the payload, Manfred Okum for designing and producing the Yagi antenna, Fahmi Rouatbi for help with the Ellipse-D and the UAV, Torsten Fiolka for help with the UAV and hardware recommendations, Matthias Mandt and Benjamin Knödler for help with RF measurements and analyses, Jannik Springer for support with an RF signal generator, and Ulrich Engel and Lars Brötje for help with the GPS. I also had the opportunity to supervise the master theses of

Markus Krestel and Thore Gerlach, which helped me by enlightening other aspects of the sensor path planning problem discussed in this thesis.

I also would like to thank all those who read drafts of this thesis and provided valuable feedback and helpful discussions: Lars Borutzky, André Brandenburger, Snezhana Jovanoska, Laura Over, Hans Schily, Isabel Schlangen, and Christoph Vollweiter.

Finally, I would like to thank my girlfriend Marie and my family for their support during the time of writing this thesis.

# Impact Statement

The increasing availability of UAVs offers the option to localize RF emitters with small airborne systems. Path planning for those systems is of high importance, as the path taken by the UAV influences the localization performance. Several works in the literature consider path planning for such systems. But these works either do not take the inherent uncertainty of this problem into account or do not minimize the time until the emitter is localized, which in many applications is an important metric. The research in this thesis closes this gap. An algorithm is developed that leads to a faster time until the target is localized than algorithms from the literature.

The produced algorithm could be used in several important real world applications. For example, consider a *search and rescue* task. Persons that ski in areas at risk of avalanches can carry a special type of RF emitter, called an avalanche transceiver. If an avalanche occurs, rescue teams can search for these emitters. An emitter-localizing UAV that uses a path optimized to reduce the localization time, leads to a shorter time until the transceiver is localized. This leads to a shorter time of the person buried under the avalanche and therefore to a higher chance of survival.

In *animal migration research*, migration studies are often performed by tagging animals with small RF emitters. By localizing these emitters in regular time intervals, the movement of the animals can be deduced. If researchers can use a UAV that localizes the animals in shorter time, they either would need to spent less time in the field or could survey greater populations.

In the military task of *electronic intelligence*, an important task is to localize hostile radar stations, for example for suppression of enemy air defences (SEAD).

To counter localization, radars are not always transmitting, making a quick localization important while they are active. Commonly, airborne electronic intelligence is performed by expensive fixed-wing aircraft. Using small inexpensive UAVs would provide a cheaper solution to the localization task.

The *localization of electromagnetic interference* is an important civilian task. Defect electronic devices can emit RF waves that interfere with other usages of the electromagnetic spectrum such as communication. Finding the source of the interference quickly is important to minimize the disruption. A UAV equipped with the developed algorithm could be used to find such a source quickly.

In these applications, the developed algorithm would provide a path that localizes the RF emitter faster, leading to tangible improvements in the execution of the task.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| API | Application Programming Interface. 167, 183 |
| BFGS | Broyden-Fletcher-Goldfarb-Shanno algorithm. 12–15, 78, 79, 81, 134, 137–139, 144, 149, 155, 157, 189 |
| CPU | Central Processing Unit. 118, 172 |
| CRLB | Cramér-Rao Lower Bound. 57, 61 |
| CRN | Common Random Number. 122, 124, 128, 129, 132, 134, 139, 140, 142, 143, 145, 148, 150, 156 |
| DF | Direction Finding. 3, 4, 41, 62, 63, 72, 77, 78, 81–83 |
| DOA | Direction of Arrival. 3, 62, 63 |
| EKF | Extended Kalman Filter. 50, 161 |
| ENTPP | Entropy-based Path Planner. 109, 110, 112, 114–116, 147, 152, 154, 155, 162 |
| ENTPP-8 | Entropy-based Path Planner with eight samples. 109, 110, 112, 114–116 |
| ENU | East North Up. 175 |
| FIM | Fisher Information Matrix. 57, 61 |
| FOV | Field of View. 77, 78 |
| GPS | Global Positioning System. 42, 167, 169, 175, 181 |
| HMM | Hidden Markov Model. 43 |
| ID | Identifier. 161 |
| IEKF | Iterated Extended Kalman Filter. 50, 54, 55 |
| INS | Inertial Navigation System. 42, 169, 172, 182 |
| IP | Internet Protocol. 172 |

JSAT Java Statistical Analysis Tool. 15, 138

L-BFGS-B Limited-memory BFGS with Bounds. 157
LTE Long Term Evolution. 167, 169, 172, 173

MAB Multi-armed Bandit. 18, 19
MC Monte Carlo. 117, 147, 153, 154
MCTS Monte Carlo Tree Search. 126, 127
MDP Markov Decision Process. 21–27, 29–32, 34, 35, 37–39, 44
MPC Model Predictive Control. 29, 30
MSE Mean Squared Error. 59, 60, 62

NBO Nominal Belief-state Optimization. 74

OCBA Optimal Computing Budget Allocation. 126

PID Proportional Integral Derivative. 162
PMC Plain Monte Carlo. 122, 128, 139, 142, 145, 148, 156
POMDP Partially Observable Markov Decision Process. 21, 35, 37–39, 75, 76, 93

RAM Random Access Memory. 172
RF Radio Frequency. iii, vii, viii, 3, 40, 68, 77, 85, 118, 160, 161, 164, 167, 168, 185
RMSE Root Mean Squared Error. 59, 60, 72, 93, 97–100, 178, 184
ROS Robot Operating System. 173, 174
RSSI Received Signal Strength Indicator. 4, 160–163

SARSA State-Action-Reward-State-Action. 35
SEAD Suppression of Enemy Air Defences. vii
SGD Stochastic Gradient Descent. 16, 17, 144, 148, 149
SOPT Sequential Optimal Localization of Pseudo Targets. 109, 110, 112, 114–116, 118, 163

UAV Unmanned Aerial Vehicle. iii, vii, viii, 2, 4, 5, 7, 75, 85, 86, 91, 159–163, 165–167, 169, 172–176, 178, 180, 182, 184, 185, 189, 190, 192

UCB Upper Confidence Bound. 126
UCL University College London. 8, 9
UCT Upper Confidence Bounds applied to Trees. 126
USB Universal Serial Bus. 172
USRP Universal Software Radio Peripheral. 169

VPN Virtual Private Network. 172

# List of Symbols

| | |
|---|---|
| $x$ | Lower case italics is either a scalar value, or a function. |
| $X$ | Upper case italics indicates an integer or a point |
| $\mathbf{x}$ | Lower case bold indicates a column vector. |
| $\mathbf{X}$ | Upper case bold indicates a matrix. |
| X | Upper case nonitalics indicates a random variable. |
| $\mathbb{X}$ | Double lines indicates a finite set, except the well known symbols $\mathbb{N}$ and $\mathbb{R}$ for the natural and real numbers. |
| $\mathcal{X}$ | Calligraphic symbols indicate either a nonfinite set or a probability related symbol, for example the normal distribution $\mathcal{N}$. |
| $\mathbf{X}^{\mathrm{T}}$ | Nonitalic superscript T indicates matrix or vector transpose. |
| $\mathbf{x}^{s}$ | Other superscripts indicate an owner or a specialization of the variable. Most important are $s$ for the sensor, $t$ for the target, $b$ for the belief and $B$ for the base policy. |
| $\tilde{\mathbf{x}}$ | Point estimate |
| $\mathbf{x}_k$ | Subscripts indicate an index. Different values of the index refer to different values. |
| $\mathbf{x}_{ka}$ | Multiple indices are not comma separated. |
| $\mathbf{x}_{[k+1]a}$ | Index shifts with multiple indices use square brackets. Here the indices are $k+1$ and $a$. |
| $\mathbf{x}_{1:R}$ | A colon notates a range of indices, forming a vector with $\mathbf{x}_{1:R} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_R)$ |
| $\mathbf{x}_{k[1:R]}$ | Ranges with multiple indices use square brackets. Here $\mathbf{x}_{k[1:R]} = (\mathbf{x}_{k1}, \mathbf{x}_{k2}, ..., \mathbf{x}_{kR})$ |
| $\overline{x}$ | Upper limit on $x$. |
| $\underline{x}$ | Lower limit on $x$. |

---

| | |
|---|---|
| $\mathcal{A}$ | Action space. 70, 92 |
| $\mathcal{B}$ | Belief space. 37, 59 |

| | |
|---|---|
| $\mathcal{C}^{95}$ | Confidence region. 165 |
| $\mathcal{C}^t$ | Possible target locations. 95 |
| $\mathcal{E}$ | Confidence ellipsoid. 51, 57, 61 |
| $\mathcal{H}^D$ | Differential entropy. 37, 60 |
| $\mathcal{H}^S$ | Discrete entropy or Shannon entropy. 60, 109 |
| $\mathcal{L}$ | Likelihood. 45 |
| $\mathcal{N}$ | Multivariate normal distribution. 48 |
| $\mathcal{P}$ | Probability of an event. 60 |
| $\mathcal{T}^B$ | Set of termination beliefs. 38, 93 |
| $\mathcal{T}^X$ | Set of termination states. 24 |
| $\mathcal{X}$ | State space. 22, 91 |
| $\mathcal{X}^m$ | Measurement locations close to the belief. 103 |
| $\mathcal{X}^s$ | Sensor state space. 91, 135 |
| $\mathcal{X}^t$ | Target state space. 40, 91 |
| $\mathcal{Z}$ | Measurement space. 35, 40 |
| | |
| $\mathbb{A}$ | Discretized action space. 27, 105, 165 |
| $\mathbb{B}$ | Set of beliefs. 138 |
| $\mathbb{I}$ | Set of grid cell indices. 54 |
| $\mathbb{N}$ | Natural numbers including zero. 22 |
| $\mathbb{S}$ | Set of target position samples. 132 |
| $\mathbb{W}^p$ | Set of possible process noises. 27 |
| $\mathbb{X}$ | Finite state space. 27, 60 |
| $\mathbb{Z}$ | Set of all integers. 128 |
| | |
| $\beta$ | Angle between measurement locations. 66 |
| $\Gamma$ | Gamma function. 61 |
| $\gamma$ | Discount. 23, 25 |
| $\Delta t$ | Time difference. 22 |
| $\Delta x^b$ | Cell width of the belief grid. 52 |
| $\Delta y^b$ | Cell height of the belief grid. 52 |
| $\delta$ | Dirac delta function. 37 |
| $\delta x^b$ | Width of the belief grid. 104 |
| $\delta y^b$ | Height of the belief grid. 104 |
| $\varepsilon$ | Small positive value. 12 |
| $\zeta$ | Cosine similarity. 166 |
| $\underline{\zeta}$ | Cosine similarity threshold. 167, 176 |
| $\eta$ | Step size of a gradient-based algorithm. 13, 135 |
| $\theta$ | Target bearing. 63 |
| $\lambda$ | Penalty parameter in the direction finder. 173 |
| $\lambda^e$ | Eigenvalue. 62 |
| $\xi$ | Normalization in the grid-based Bayes filter. 53 |
| $\pi$ | Policy. 24, 37 |
| $\pi^B$ | Base policy. 31, 98 |
| $\pi^*$ | Optimal policy. 94 |

| | |
|---|---|
| $\mu$ | Expected RMSE. 60, 93 |
| $\overline{\mu}$ | Localization accuracy threshold. 93, 108, 176 |
| $\sigma$ | Measurement standard deviation. 64, 92, 108, 176 |
| $\phi$ | Parameter vector. 35 |
| $\boldsymbol{\psi}$ | Shifted antenna pattern. 166 |
| $\omega$ | Sample path. 101, 121, 127 |
| $\omega^m$ | Measurement noise seed. 101, 127 |
| $\omega^x$ | X-position of the target in a sample path. 101, 127 |
| $\omega^y$ | Y-position of the target in a sample path. 101, 127 |
| | |
| $K$ | Finite horizon. 23, 87 |
| $L$ | Number of iterations. 19, 133, 137, 144 |
| $M$ | Number of Monte Carlo runs. 140 |
| $N$ | Computational budget. 18, 130, 141, 176 |
| $P$ | Arbitrary dimension of a space. 12, 22 |
| $R$ | Rollouts per action. 101, 131, 133, 138 |
| $S$ | Dimension of the sensor state space. 40 |
| $T$ | Dimension of the target state space. 40 |
| $X^a$ | X-dimension size of the action grid. 105, 153 |
| $X^b$ | X-dimension size of the belief grid. 52 |
| $Y^a$ | Y-dimension size of the action grid. 105, 153 |
| $Y^b$ | Y-dimension size of the belief grid. 52 |
| $Z$ | Dimension of the measurement space. 35, 40 |
| | |
| $\mathbf{A}$ | Measurement matrix for the direction finder. 173 |
| $\mathbf{B}$ | Approximation of the Hessian. 14 |
| $\mathbf{F}$ | Transition matrix of the process model. 48 |
| $\mathbf{H}$ | Measurement matrix. 48 |
| $\mathbf{I}$ | Identity matrix. 15 |
| $\mathbf{J}$ | Fisher information matrix. 57, 65 |
| $\mathbf{K}$ | Kalman gain. 49 |
| $\mathbf{M}$ | Weighting matrix for the BFGS derivation. 14 |
| $\tilde{\mathbf{P}}^t$ | Point estimate covariance. 48, 54, 61, 97, 139 |
| $\mathbf{Q}$ | Process noise covariance. 48 |
| $\mathbf{R}$ | Measurement noise covariance. 43 |
| $\mathbf{S}$ | Innovation covariance. 49 |
| | |
| B | Belief. 94 |
| E | Point estimator of the target state. 57 |
| F | Random sample of the function $f$. 16 |
| G | Random sample of the gradient. 16, 135 |
| K | Decision step when the MDP terminates. 24, 34 |

| | |
|---|---|
| R | Rollouts per action. 139 |
| S | Score. 56 |
| $\mathrm{W}^m$ | Measurement noise. 94 |
| $\mathrm{W}^p$ | Process noise. 22 |
| $\mathrm{X}^s$ | Sensor state. 94 |
| $\mathrm{X}^t$ | Target state. 44, 94 |
| | |
| $a^{\mathrm{FD}}$ | Finite difference size in one direction. 137 |
| $b$ | Belief about the joint state. 70 |
| $b^t$ | Belief about the target state. 44 |
| $c$ | Cost function. 92 |
| $c^S$ | Surrogate cost function. 29 |
| $d^M$ | Mahalanobis distance. 50 |
| $d^o$ | Mean distance to the optimum. 140 |
| $e$ | Euler's number. 61 |
| $f$ | A function. 11 |
| $f^a$ | MDP transition function. 22, 92 |
| $f^d$ | Deterministic transition function. 29 |
| $f^\pi$ | State transition according to a policy. 25 |
| $f^s$ | Sensor transition function. 70, 92 |
| $h$ | Measurement function. 35, 43 |
| $h^{\mathrm{DF}}$ | Measurement function for a DF sensor. 64, 91 |
| $k$ | Decision step. 22 |
| $l$ | Iteration. 13, 31, 132, 135, 137 |
| $m$ | Monte Carlo run. 140 |
| $q^B$ | Action value of the base policy. 31 |
| $\hat{q}^B$ | Rollout result for a specific sample path. 101 |
| $q^*$ | Optimal action value function. 26 |
| $q^\pi$ | Action value function. 25 |
| $r$ | A distance. 64 |
| $\underline{r}$ | Minimal distance to the confidence region. 176 |
| $r^c$ | Pearson correlation coefficient. 149 |
| $t$ | Time. 22 |
| $t^d$ | Time cost due to travel distance. 92 |
| $t^m$ | Measurement duration. 92, 108, 176, 184 |
| $u$ | Quadratic function approximation. 14 |
| $v^B$ | Value function of the base policy $\pi^B$. 33 |
| $v^*$ | Optimal value function. 26 |
| $v^\pi$ | Value function of policy $\pi$. 25 |
| $v^R$ | Value function of the rollout policy $\pi^R$. 33 |
| $v^s$ | Speed of the UAV. 79, 92, 108, 176 |
| $\overline{v}^s$ | Maximal speed of the UAV. 184 |
| $w^m$ | Scalar measurement noise. 64, 91 |
| $\underline{x}^b$ | Lower x coordinate of the belief grid. 52, 96 |
| $\overline{x}^b$ | Upper x coordinate of the belief grid. 52, 96 |

| | |
|---|---|
| $x^s$ | X position of the sensor. 91 |
| $x^t$ | X position of the target. 91 |
| $\underline{y}^b$ | Lower y coordinate of the belief grid. 52, 96 |
| $\overline{y}^b$ | Upper y coordinate of the belief grid. 52, 96 |
| $y^s$ | Y position of the sensor. 91 |
| $y^t$ | Y position of the target. 91 |
| $z$ | Scalar bearing measurement. 45, 46, 64, 91 |
| | |
| $\mathbf{0}$ | Zero vector. 43 |
| $\mathbf{a}$ | Action. 22, 70, 92, 105 |
| $\mathbf{a}^*$ | Optimal action. 121, 129, 139 |
| $\mathbf{i}$ | Information vector. 36 |
| $\mathbf{m}$ | Measured power vector. 166, 173 |
| $\mathbf{p}$ | Line search direction. 15 |
| $\mathbf{r}^*$ | Optimal signal reconstruction in the DF. 173 |
| $\mathbf{w}^m$ | Measurement noise. 35, 43 |
| $\mathbf{w}^p$ | Process noise. 27, 44, 48 |
| $\mathbf{x}$ | Joint state. 91 |
| $\mathbf{x}^s$ | Sensor state. 64, 79, 91 |
| $\mathbf{x}^t$ | Target state. 91 |
| $\tilde{\mathbf{x}}^t$ | Point estimate of the target state. 48, 53, 97, 139 |
| $\tilde{\mathbf{x}}^{t,ev}$ | Expected value of the target state. 53, 97 |
| $\tilde{\mathbf{x}}^{t,map}$ | Maximum a-posteriori estimate of the target state. 53, 97 |
| $\mathbf{y}$ | Innovation. 49 |
| $\mathbf{z}$ | Measurement. 35, 40, 43 |
| | |
| update | Belief update. 45, 94, 109 |

# Chapter 1

# Introduction

Emitter localization plays an important role in many military and civilian contexts, such as finding the position of hostile radars [Wiley, 2006, Ch. 6], studying animals by radio-tagging [Fuller and Fuller, 2012], or search and rescue [McClung and Schaerer, 2006, Ch. 9]. In those applications, an object emits a signal and its position needs to be found. The object is commonly called a *target* and could be a radar, an animal tagged with a radio beacon, an emergency locator beacon, or any other emitter of interest.

A sensor cannot measure the position of the target directly, but only the signal strength, direction, and frequency of the received signal. By combining multiple of those measurements, the target position can be determined. The position of a target and other quantities of interest, like the velocity of a moving target, are called the *target state*. Methods to combine measurements and form an estimate of the target state are studied in the field of *sensor data fusion* [Bar-Shalom et al., 2001; Koch, 2014]. Typically, those methods are based on Bayesian statistics and use statistical models of the measurement process and the behaviour of the target. These models are used to form a probability distribution over the true target state, which is called a *belief* [Thrun et al., 2005, Ch. 1].

Under a *sensor system* we understand the combination of one or multiple sensors, their sensor data fusion algorithms, as well as one or more platforms on which these sensors are mounted. At a given time, a sensor system produces a single belief about the target state, based on the previously received measurements. We refer to

the objective of a sensor system as its *sensing task*. Exemplary sensing tasks are to find all targets in a specific region, track one or multiple targets, or localize a target. A sensing task is specified together with metrics, which should be optimized, such as the accuracy of a localization or the time until all targets in a region are found.

There are several complementary and interrelated ways to improve the ability of a sensor system to perform a sensing task. The first would be to upgrade the hardware. This can be done by technological progress, for instance by the construction of a better receiver. Another option to upgrade the hardware of a sensor system is to scale the existing sensor system, adding additional sensors to form a sensor network. The second method to improve a sensor system is by better processing of the measurements. This can be done either on the signal processing level to produce more accurate measurements from the raw sensor readings, or on the sensor data fusion level to form more accurate target tracks.

The third method for performance improvement of a sensor system is to control the sensor in an intelligent and adaptive way. This method is studied in the field of *sensor management* [Hero III and Cochran, 2011] and is also the method considered in this thesis. Sensor systems typically have several degrees of freedom, which influence the measurement process and can be controlled. Modern electronically scanned array radars, for example, have the ability to almost instantaneously change the beam direction and waveform. Sensors mounted on mobile platforms, like unmanned aerial vehicles (UAVs), can adaptively change their position. The field of sensor management studies algorithms that control these degrees of freedom, to better execute the sensing task. Those algorithms use the belief produced by the sensor data fusion algorithms, knowledge about the measurement process, as well as potential context information. This thesis focuses on *sensor path planning*, a subfield of sensor management that optimizes the position of the sensor.

## 1.1 Emitter localization

An emitter is a target that transmits a signal, for example a radio wave or acoustic signal. Emitter localization is the problem of finding the position of such targets.

In this thesis, we are interested in localizing an RF emitter. This problem appears in many different situations. Defective or wrongly configured electrical devices can emit electromagnetic interference, which are unwanted RF signals that disrupt communication via mobile phones and radio. It is therefore necessary to locate the emitter to remove this interference. In military applications, the positions of hostile radars or jammers need to be determined. In biology, the behaviour of animals is often studied by attaching small RF emitters to them and tracing their position over time. An emergency locator beacon is an RF emitter that is activated in an emergency situation, like falling overboard from a ship or becoming buried by an avalanche. In those situations, a quick localization of the emitter by a search and rescue team increases the survival chance.

Sensors used for RF emitter localization measure the received electromagnetic waves and can therefore only extract limited information. This could be the direction of arrival (DOA), the frequency, or the signal strength. Most important, the distance towards the emitter is not available if its power is unknown. Therefore, commonly DOA measurements or *bearing measurements* are used for the localization. Sensors that produce bearing measurements are called *direction finding sensors* (DF sensors).

With bearing measurements, it is required to combine multiple measurements to localize the emitter. A straightforward way to create multiple measurements is the use of multiple sensors. This is a useful approach for extremely short signals, as the measurements of multiple well-positioned DF sensors can be intersected to instantaneously localize the emitter. If the emitter is instead active over a longer time period, an alternative is to move a single sensor to a new position and take another measurement there. A mobile sensor is a common situation: when localizing the source of an electromagnetic interference, a mobile detector van is used. In animal studies, researchers often take measurements manually from different positions using directional antennas [Tokekar et al., 2013; Hui et al., 2021]. In military surveillance, DF sensors can be mounted on aircrafts to take multiple measurements during a reconnaissance flight.

Over the last decade, small UAVs have become more and more prevalent [Kumar and Michael, 2012; Floreano and Wood, 2015]. Due to their mobility and ease of use, they are well suited as a sensor platform. Typical ways to perform emitter localization with a UAV are the measurement of signal strength [Körner et al., 2010; Nguyen et al., 2019], also called *received signal strength indicator* (RSSI), and the usage of DF sensors [Vrba et al., 2019]. Due to cost, complexity, and weight requirements, often a simple way of direction finding is used: a directional antenna that is mounted on the UAV [Cliff et al., 2015; Isaacs et al., 2014; Vonehr et al., 2016]. When rotating the UAV, the received signal power peaks in the direction of the signal. Such a sensor system needs to stop movement when taking a measurement. This assumption is not considered by the majority of existing sensor path planning algorithms in the literature [Hammel et al., 1989; Oshman and Davidson, 1999; Hoffmann and Tomlin, 2010; Doğançay, 2012]. Instead they assume that there is no interruption an the measurements are taken at regular intervals. Therefore, novel path planning algorithms that take these constraints into account are required.

## 1.2 Statement of the objective

When localizing an emitter with bearing measurements, the relative position of sensor and target has a strong influence on the accuracy of the localization. This fact motivates a large body of research in sensor path planning. Most sensor path planning algorithms assume a generation of measurements in regular time intervals. Direction finding based on array antennas is almost instantaneous [Wiley, 2006, Ch. 5]. In this case, modelling the measurement generation in regular time steps is a valid assumption. However, due to cost, complexity, and weight requirements, small UAV-based sensor systems often use simpler sensors like a rotating directional antenna [Cliff et al., 2015; Vrba et al., 2019; Isaacs et al., 2014; Vonehr et al., 2016]. Such a sensor leads to a trade-off between taking a measurement and moving the platform, which influences the optimal sensor path. To use these sensor systems optimally, sensor path planning needs to be adapted to this trade-off.

There are two interlinked sources of uncertainty in the problem of emitter localization. When making the decision for the next measurement location, only an uncertain belief about the target position is available. In addition, the values of future measurements are uncertain. To perform an optimal decision, these uncertainties need to be taken into account.

A successful emitter localization is the result of a sequence of multiple measurements. These measurements need to be taken with different relative positions of target and sensor, called *sensor-to-target geometries*. As changing the measurement location means to physically move the UAV, this does not only change the sensor-to-target geometry of the next measurement, but also influences the sensor position for future measurements. While the sensor-to-target geometry can be changed for those future measurements again, this leads to additional time spent travelling. Therefore, it is important to take future measurements into account when deciding on the next measurement location.

An algorithm that takes multiple future measurement locations into account is called a *nonmyopic* path planner. These algorithms can also be distinguished in how far they look into the future, which is called the *planning horizon* of the algorithm. In contrast, a *myopic* path planner only considers the immediate next measurement location.

Literature on the problem of sensor path planning for emitter localization with a small UAV exists, however, the presented path planners are typically either myopic [Vander Hook et al., 2014; Cliff et al., 2015; Dressel and Kochenderfer, 2018] or with a limited planning horizon [Tokekar et al., 2011]. Additionally, the uncertainty in the problem is often not fully taken into account [Cliff et al., 2015; Dressel and Kochenderfer, 2018; Vander Hook et al., 2015]. Therefore, a novel path planner for this problem is developed in this thesis. In total, this thesis makes the following contributions:

**Create a novel path planner for localizing an emitter with a UAV.** A novel path planner is developed for the emitter localization problem with a trade-off between taking a measurement and moving the platform. The path planner takes

into account the uncertainty present in the problem and has a nonmyopic planning horizon.

**Optimize the action selection in a policy rollout algorithm.** The path planner is based on the policy rollout principle from approximate dynamic programming. Action selection in policy rollout is often done in an inefficient way. More effective ways to perform the action selection are examined in this thesis.

**Analyse the relationship between action selection and total received cost.** It is not guaranteed that improving the action selection in a policy rollout, which means finding an action with a better action value, results in a better performance of the resulting algorithm. The correlation between minimizing the action value and the resulting time until the localization of the target is analysed.

**Perform a demonstration of the novel path planner in an experimental sensor system.** The resulting path planner is tested in an experimental sensor system. Differences between this sensor system and the assumptions of the path planner are analysed.

## 1.3 Structure of the thesis

This thesis covers the area of sensor management, which brings together optimization and sensor data fusion. Chapter 2 gives a brief overview over optimization techniques. The focus is put on two problems: Function minimization and Markov decision processes. In function minimization, an important topic is minimizing functions of which only stochastic samples are available. Markov decision processes are sequential decision problems. They are useful to model problems where actions have influences over multiple decision steps. Several solution methods are presented, especially the policy rollout algorithm which is used in this thesis.

Chapter 3 discusses three important sensor related basics of this thesis. First, the Bayesian formalism of sensor data fusion is introduced, as well as ways to

model the uncertainty of the belief. In the second section, direction finding sensors are introduced. The Fisher information, introduced in the first section, is computed for direction finding sensors and optimal sensor-to-target geometries are described. It will be seen that the influence of the sensor-to-target geometry on the localization accuracy is significant. The last section covers sensor management, with a special focus on sensor path planning.

The first contribution of this thesis is described in Chapter 4. Here, a path planner to localize a target with a UAV is developed. In contrast to path planners from the literature, the algorithm of this thesis performs a nonmyopic planning of the sensor path and takes into account different future measurement realizations, based on the uncertainty of the current state estimate. Simulations show that the path planner localizes targets faster than comparable algorithms from the literature.

The path planner is further improved in Chapter 5. This chapter is based on the idea that action selection in the policy rollout algorithm can be interpreted as the minimization of a stochastic function. Several algorithms to solve this task are compared and the resulting trade-off between computational budget and performance is analysed. The chapter also shows that the optimum of the rollout action value correlates with the actual performance, motivating the importance of optimizing the action selection.

In Chapter 6 the path planner of this thesis is evaluated in an experimental sensor system. The sensor system and setup is described and several required adaptions to the algorithm are introduced. The sensor system is shown to be able to localize an emitter.

Finally, Chapter 7 presents the general conclusions and contributions of this thesis as well as possible future extensions of this work.

## 1.4 Notation

This thesis uses the following notation: A lower case italic letter such as '$x$' or '$f$' is either a scalar value or a function. An upper case italic letter such as '$X$' indicates an integer or point. Bold letters '$\mathbf{x}$' indicate a vector if lower case. If bold letters are

upper case, such as '$\mathbf{X}$', they indicate a matrix. Upper case nonitalic letters such as 'X' are random variables. Continuous sets are denoted in calligraphic such as '$\mathcal{X}$', while finite sets use double lines, such as '$\mathbb{X}$'. Calligraphic notation is also used for terms of probability theory, such as the likelihood $\mathcal{L}$. An overview over the notation can be found in the list of symbols at the beginning of this thesis.

Whether a variable is denoted as a random variable is seen from the perspective of the path planner. When the path planner is executed at decision step $k$, all prior measurements $\mathbf{z}_0, ..., \mathbf{z}_k$ are known. In this case no distinction between the random variable and its instantiation is made. The distinction between a random variable and its instantiation is made for all values not known to the path planner at decision step $k$, which are for example future measurements $\mathrm{Z}_{k+1}, \mathrm{Z}_{k+2}$ or the true target position X.

## 1.5 List of own publications

Publications that were written before enrolment at UCL:

1. Folker Hoffmann and Alexander Charlish. A resource allocation model for the radar search function. In *International Radar Conference*, pages 1–6, Lille, France, 2014. IEEE

2. Alexander Charlish and Folker Hoffmann. Anticipation in cognitive radar using stochastic control. In *2015 IEEE Radar Conference (RadarCon)*, pages 1692–1697, Arlington, Virginia, USA., 2015. IEEE

3. Folker Hoffmann, Alexander Charlish, and Wolfgang Koch. Trajectory optimization for multi-platform bearing-only tracking with ghosts. In *Proceedings of the 19th International Conference on Information Fusion (FUSION)*, pages 39 – 44, Heidelberg, Germany, 2016a

4. Folker Hoffmann, Matthew Ritchie, Francesco Fioranelli, Alexander Charlish, and Hugh Griffiths. Micro-doppler based detection and tracking of UAVs with multistatic radar. In *IEEE Radar Conference (RadarConf)*, pages 893–898, Philadelphia, PA, USA, 2016b. IEEE

5. Colin Horne, Matthew Ritchie, Hugh Griffiths, Folker Hoffmann, and Alexander Charlish. Experimental validation of cognitive radar anticipation using stochastic control. In *Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, USA, 2016

6. Alexander Charlish and Folker Hoffmann. Cognitive radar management. In *Novel Radar Techniques and Applications Volume 2: Waveform Diversity and Cognitive Radar, and Target Tracking and Data Fusion*, chapter 5, pages 157–193. Institution of Engineering and Technology, 1 edition, 2017

Publications that were written during enrolment at UCL:

7. Markus Krestel, Folker Hoffmann, Hans Schily, Alexander Charlish, and Sven Rau. Passive emitter direction finding using a single antenna and compressed sensing. In *2019 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, pages 1–5, Bonn, Germany, 2019. IEEE

8. Folker Hoffmann, Hans Schily, Alexander Charlish, Matthew Ritchie, and Hugh Griffiths. A rollout based path planner for emitter localization. In *Proceedings of the 22nd International Conference on Information Fusion (FUSION)*, Ottawa, ON, Canada, 2019

9. Alexander Charlish, Folker Hoffmann, Christoph Degen, and Isabel Schlangen. The development from adaptive to cognitive radar resource management. *IEEE Aerospace and Electronic Systems Magazine*, 35(6):8–19, June 2020

10. Folker Hoffmann, Alexander Charlish, Matthew Ritchie, and Hugh Griffiths. Sensor path planning using reinforcement learning. In *Proceedings of the 23nd International Conference on Information Fusion (FUSION)*, Rustenburg, South Africa (Virtual), 2020

11. Folker Hoffmann, Alexander Charlish, Matthew Ritchie, and Hugh Griffiths. Policy rollout action selection in continuous domains for sensor path plan-

ning. *IEEE Transactions on Aerospace and Electronic Systems*, pages 2247–2264, 2021

12. Hans Schily, Folker Hoffmann, and Alexander Charlish. A comparison of distributed and centralized control for bearing only emitter localization with sensor swarms. In *SCI-341: Situation Awareness of Swarms and Autonomous Systems*, Tallinn, Estonia (Virtual), 2021

13. André Brandenburger, Folker Hoffmann, and Alexander Charlish. Co-training an observer and an evading target. In *24th International Conference on Information Fusion (FUSION)*, Rustenburg, South Africa, 2021

14. Thore Gerlach, Folker Hoffmann, and Alexander Charlish. Policy rollout action selection with knowledge gradient for sensor path planning. In *24th International Conference on Information Fusion (FUSION)*, Rustenburg, South Africa, 2021

# Chapter 2

# Optimization and Approximate Dynamic Programming

This thesis considers sensor management, which is optimizing the control of a sensor system to achieve a sensing task. Algorithms for sensor management are often based on mathematical optimization. In this chapter, we review the basics of optimization, focused on two topics: function minimization and Markov decision processes. In function minimization, we discuss gradient-based optimization, stochastic gradient descent and multi-armed bandits. Markov decision processes are a way to model problems that consist of sequential decisions where each decision influences the state in a random way.

## 2.1 Function minimization

In this section, the minimization of a function $f$ is discussed. The function $f$ is called the *objective function* and is used to specify the problem that should be solved. We consider three cases of objective functions that differ in their domain and in whether the output is directly observable.

The first case assumes a differentiable function

$$f : \mathbb{R}^P \to \mathbb{R} \tag{2.1}$$

with gradient $\nabla f(\mathbf{a})$ at point $\mathbf{a} \in \mathbb{R}^P$. The goal is to find a minimum $\mathbf{a}^* \in \mathbb{R}^P$. If $f(\mathbf{a}^*) \leq f(\mathbf{a})$ for all $\mathbf{a} \in \mathbb{R}^P$, $\mathbf{a}^*$ is called a global minimum. Many methods only find a local minimum, which means that

$$f(\mathbf{a}^*) \leq f(\mathbf{a}) \text{ for all } \mathbf{a} \text{ with } \|\mathbf{a}^* - \mathbf{a}\|_2 < \varepsilon \tag{2.2}$$

for an $\varepsilon > 0$. In high-dimensional problems also *saddle points* become increasingly a problem for gradient-based methods [Goodfellow et al., 2016, Ch. 8]. In those points the gradient of $f$ is zero, but the point is neither a local nor a global minimum. We present gradient descent, Newton's method, and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm as algorithms that search for a local minimum of $f$.

The second case assumes the same basic setting, but the value of $f$ is not observable. Instead, random variables $\mathrm{F}, \mathrm{G}$ are available, with

$$\mathbb{E}[\mathrm{F}(\mathbf{a})] = f(\mathbf{a}) \tag{2.3}$$

$$\mathbb{E}[\mathrm{G}(\mathbf{a})] = \nabla f(\mathbf{a}) \,. \tag{2.4}$$

Those can be interpreted as a noisy sample of the function and its gradient. In this case, the method of stochastic gradient descent can be applied.

The third case assumes a function $f : \mathbb{A} \to \mathbb{R}$, where $\mathbb{A}$ is a finite set. Similar to the second case, only noisy samples F of the function are available with

$$\mathbb{E}[\mathrm{F}(\mathbf{a})] = f(\mathbf{a}) \,. \tag{2.5}$$

As the set $\mathbb{A}$ is unordered, the concept of a local minimum is not useful and the goal is to find the global minimum $\mathbf{a}^*$, with $f(\mathbf{a}^*) \leq f(\mathbf{a})$ for all $\mathbf{a} \in \mathbb{A}$. We will present the algorithm of sequential halving for this problem.

### 2.1.1 Differentiable functions

There exist a large number of minimization algorithms for differentiable functions [Nocedal and Wright, 2006]. Numerical minimization algorithms typically start

with an initial point $\mathbf{a}_1$ and iteratively update the current point $\mathbf{a}_l$ to arrive at a new point $\mathbf{a}_{l+1}$. A simple algorithm is *gradient descent*

$$\mathbf{a}_{l+1} = \mathbf{a}_l - \eta_l \cdot \nabla f(\mathbf{a}_l) \tag{2.6}$$

which selects the new point in the opposite direction of the gradient $\nabla f(\mathbf{a}_l)$, computed at the last point. This direction is the direction of steepest descent. Here $l$ is the index of the iteration and $\eta_l$ the *step size* of the algorithm. The step size might vary in each iteration or could be kept constant.

A main difficulty with gradient descent algorithms is that the convergence can be slow [Nocedal and Wright, 2006, Ch. 3]. A faster algorithm is *Newton's method*. It uses the additional information of the Hessian $\nabla^2 f$, to define the update

$$\mathbf{a}_{l+1} = \mathbf{a}_l - \eta_l \cdot \nabla^2 f(\mathbf{a}_l)^{-1} \nabla f(\mathbf{a}_l) \; . \tag{2.7}$$

In contrast to gradient descent, in Newton's method an ideal step size of $\eta_l = 1$ exists. Geometrically, Newton's method can be interpreted as approximating $f$ locally at $\mathbf{a}_l$ with a quadratic function. The point $\mathbf{a}_l - \nabla^2 f(\mathbf{a}_l)^{-1} \nabla f(\mathbf{a}_l)$ is the minimum of this quadratic function, which motivates the ideal step size of one. However, this step size is only useful when the quadratic function approximation is valid for a sufficiently large radius around the current point. Otherwise, smaller step sizes need to be used. The algorithm assumes that the Hessian is positive definite and care must be taken when this is not the case [Nocedal and Wright, 2006, Ch. 3].

Another disadvantage of Newton's method is that the Hessian must be computed explicitly. So called *quasi-Newton methods* do not require the Hessian, but instead try to approximate it using only information from the gradient. The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is such a quasi-Newton algorithm. We now sketch the ideas of the BFGS algorithm and refer for more details to [Nocedal

and Wright, 2006, Ch. 6] and [Dennis, Jr. and Moré, 1977]. Similar as in Newton's method, the BFGS update consists of

$$\mathbf{a}_{l+1} = \mathbf{a}_l - \eta_l \cdot \mathbf{B}_l^{-1} \nabla f(\mathbf{a}_l) \tag{2.8}$$

where $\mathbf{B}_l$ is an approximation of the Hessian $\nabla^2 f(\mathbf{a}_l)$. After the new point $\mathbf{a}_{l+1}$ is selected, this approximation is updated. There are two major ideas in the update of $\mathbf{B}_l$. First, the gradient of the quadratic approximation

$$u(\mathbf{s}) = f(\mathbf{a}_{l+1}) + \nabla f(\mathbf{a}_{l+1})^{\mathrm{T}} \mathbf{s} + \frac{1}{2} \mathbf{s}^{\mathrm{T}} \mathbf{B}_{l+1} \mathbf{s} \tag{2.9}$$

at $\mathbf{a}_{l+1}$ should match the gradient of the function $f$, at the current and last point. This means

$$\nabla u(\mathbf{0}) = \nabla f(\mathbf{a}_{l+1}) \tag{2.10}$$

$$\nabla u(-\mathbf{s}_l) = \nabla f(\mathbf{a}_l) \tag{2.11}$$

where $\mathbf{s}_l = \mathbf{a}_{l+1} - \mathbf{a}_l$ is the difference between the last and the current point. Equation (2.10) is satisfied due to the definition of the quadratic approximation, while (2.11) is a constraint on $\mathbf{B}_{l+1}$. This constraint alone does not make $\mathbf{B}_{l+1}$ unique. The idea of the BFGS algorithm is that the approximation of the Hessian should not change much between successive iterations. Therefore, $\mathbf{B}_{l+1}$ is defined as the symmetric, positive definite matrix which fulfils the constraint (2.11) and minimizes

$$\min_{\mathbf{B}} = \|\mathbf{M}(\mathbf{B}^{-1} - \mathbf{B}_l^{-1})\mathbf{M}\|_F \ . \tag{2.12}$$

This means the inverse of $\mathbf{B}$ should change the least between iterations if measured with the Frobenius norm weighted by a matrix $\mathbf{M}$. This weighting matrix $\mathbf{M}$ can be any symmetric nonsingular matrix which satisfies the constraint

$$\mathbf{M}^2 \mathbf{s}_l = \mathbf{g}_l \tag{2.13}$$

where $\mathbf{g}_l = \nabla f(\mathbf{a}_{l+1}) - \nabla f(\mathbf{a}_l)$ is the difference between the gradients in the last iteration [Dennis, Jr. and Moré, 1977]. The matrix that minimizes (2.12) can be computed analytically [Dennis, Jr. and Moré, 1977; Nocedal and Wright, 2006, Ch. 5]. It is possible to directly compute the inverse

$$\mathbf{B}_{l+1}^{-1} = \left(\mathbf{I} - \frac{\mathbf{s}_l \mathbf{g}_l^{\mathrm{T}}}{\mathbf{g}_l^{\mathrm{T}} \mathbf{s}_l}\right) \mathbf{B}_l^{-1} \left(\mathbf{I} - \frac{\mathbf{g}_l \mathbf{s}_l^{\mathrm{T}}}{\mathbf{g}_l^{\mathrm{T}} \mathbf{s}_l}\right) + \frac{\mathbf{s}_l \mathbf{s}_l^{\mathrm{T}}}{\mathbf{g}_l^{\mathrm{T}} \mathbf{s}_l} \tag{2.14}$$

where $\mathbf{I}$ refers to the identity matrix. Computing the inverse directly is advantageous as only the inverse of $\mathbf{B}_l$ is required in (2.8).

Typically, these three algorithms select the step size $\eta_l$ in an iteration by a *line search*. A line search searches for the optimal $\eta_l$ in either (2.6), (2.7), or (2.8), and the evaluated points $\mathbf{a}_{l+1}$ all lie on a single line

$$\mathbf{a}_{l+1} = \mathbf{a}_l + \eta_l \cdot \mathbf{p}_l \tag{2.15}$$

starting at $\mathbf{a}_l$. The direction of the line depends on the algorithm and would be $\mathbf{p}_l = -\nabla f(\mathbf{a}_l)$ for gradient descent, $\mathbf{p}_l = -\nabla^2 f(\mathbf{a}_l)^{-1} \nabla f(\mathbf{a}_l)$ for Newton's method, or $\mathbf{p}_l = -\mathbf{B}_l^{-1} \nabla f(\mathbf{a}_l)$ for BFGS. Several line search algorithms are possible. In the JSAT library[1] [Raff, 2017], the default line search algorithm for BFGS is *backtracking line search*. This algorithm starts with a high estimate for $\eta_l$, which is successively reduced, until the *Armijo rule*

$$f(\mathbf{a}_l + \eta_l \cdot \mathbf{p}_l) \leq f(\mathbf{a}_l) + c_1 \cdot \eta_l \cdot \mathbf{p}_l^{\mathrm{T}} \nabla f(\mathbf{a}_l) \tag{2.16}$$

is fulfilled. Here $c_1 \in (0,1)$ is a small constant. The right side of the equation is linear in the step size $\eta_l$ with a negative slope, because the directional derivative $\mathbf{p}_l^{\mathrm{T}} \nabla f(\mathbf{a}_l)$ is negative.

A step size $\eta_l$ is only accepted if the resulting function value, that means the left side of (2.16), is less than this linear bound. This condition signifies that there

[1] `https://github.com/EdwardRaff/JSAT`

should be a sufficient decrease in the function value to accept a value for $\eta_l$ during the line search.

### 2.1.2 Stochastic differentiable functions

The previous section assumed that the function value $f(\mathbf{a})$ can be computed, as well as the gradient $\nabla f(\mathbf{a})$. We now consider the case that neither of these is available exactly, but instead random variables $\mathrm{F}(\mathbf{a})$, $\mathrm{G}(\mathbf{a})$ exist with

$$\mathbb{E}[\mathrm{F}(\mathbf{a})] = f(\mathbf{a}) \tag{2.17}$$

$$\mathbb{E}[\mathrm{G}(\mathbf{a})] = \nabla f(\mathbf{a}) \ . \tag{2.18}$$

Effectively, this means that F is a random sample of the function value and G is a random sample of the gradient. Such a situation can appear if $f$ is not computable directly, but approximated using Monte Carlo sampling. This setting is also very common in machine learning, where an objective function - here called *loss function* - and its gradient is typically only computed on a subset, or *mini-batch*, of the training data set [Goodfellow et al., 2016, Ch. 8].

The idea of *stochastic gradient descent* (SGD) is to perform gradient descent on $f$, but instead of using the exact gradient $\nabla f(\mathbf{a})$, a random sample $\mathrm{G}(\mathbf{a})$ is used. Similar to (2.6), in each iteration a new point is derived by

$$\mathrm{A}_{l+1} = \mathrm{A}_l - \eta_l \cdot \mathrm{G}(\mathrm{A}_l) \ . \tag{2.19}$$

Note that the points $\mathrm{A}_l$ are a random sequence. It is also interesting to note that the function value F is not directly needed and the gradient sample G is sufficient. In gradient descent, the function value $f(\mathbf{a})$ is used in the line search, but performing line search with the random variable $\mathrm{F}(\mathbf{a})$ would not necessarily be helpful. In addition, a concrete realization of $\mathrm{G}(\mathbf{a})$ might not even point in the right direction of the true gradient.

The step size $\eta_l$ is very important in SGD due to the noisy gradient sample. In gradient descent, the gradient of a function becomes zero in a local minimum and therefore the iteration (2.6) becomes

$$\mathbf{a}^* = \mathbf{a}^* - \eta_l \cdot \mathbf{0} \tag{2.20}$$

for any step size $\eta_l$. The gradient of $f$ becomes smaller when closing in to a local minimum, and therefore even a constant step size might work. However, the same is not the case for SGD. As only a random sample of the gradient is available, it is likely that the realization of $\mathrm{G}(\mathbf{a}^*)$ is not zero, and therefore even if $\mathrm{a}_l$ is at a local minimum, (2.19) would move away from it.

The solution for this approach is to create a *step size schedule*. By decreasing $\eta_l$ for increasing $l$, the influence of each gradient sample becomes successively less and the algorithm converges. The major issue is to find a correct step size schedule. Theoretical convergence results exist [Goodfellow et al., 2016, Ch. 8] for a schedule that conforms to the conditions

$$\sum_{l=1}^{\infty} \eta_l = \infty \tag{2.21}$$

and

$$\sum_{l=1}^{\infty} \eta_l^2 < \infty \,. \tag{2.22}$$

The second condition ensures that the schedule eventually converges to zero, while the first condition ensures that this does not happen too fast. A solution for those conditions would be $\eta_l = 1/l$. However, these criteria do not guarantee a convergence for a fixed number of iterations and are rarely used in practice. For example in [Goodfellow et al., 2016, Ch. 8], it is recommended to reduce the step size linearly until a threshold and to keep it constant afterwards, clearly violating (2.22). An interesting overview of step size schedules can be found in [Powell, 2011, Ch.11], but focused on a nonstationary function whose true values shift over time.

### 2.1.3 Multi-armed bandits

The multi-armed bandit (MAB) problem describes the problem of a gambler facing a row of unequal slot machines ("one-armed-bandits"). When a slot machine is played, it produces a payoff or *reward*, specific to a probability distribution of this machine. The goal of the gambler is to maximize the money won, therefore he or she always wants to play the machine with the highest expected payoff. As this machine is unknown, the gambler is faced with the choice of using the machine with the empirically highest payoff (exploitation), or testing other machines (exploration), potentially finding a machine with a higher payoff. This is known as the exploration-exploitation dilemma. A common algorithm for this problem is based on upper confidence bounds [Auer et al., 2002].

A variation of this problem is the *best arm selection problem* [Bubeck et al., 2009; Audibert and Bubeck, 2010; Gabillon et al., 2012; Garivier and Kaufmann, 2016; Karnin et al., 2013]. In the standard formulation of an MAB described above, the objective is to maximize the payoff parallel to the information gathering process. In the best arm selection problem the objective is only to identify the machine with the highest payoff.

While the intuition of the problem corresponds to a maximization problem, the corresponding minimization problem is analogous. We now assume that each alternative $\mathbf{a}$ of the finite set $\mathbb{A}$ has a true cost $f(\mathbf{a})$, with

$$f : \mathbb{A} \to \mathbb{R} \tag{2.23}$$

and the minimum $\mathbf{a}^* \in \mathbb{A}$ should be found. The true cost is only observable by samples from a random variable $\mathrm{F}(\mathbf{a})$, with

$$\mathbb{E}[\mathrm{F}(\mathbf{a})] = f(\mathbf{a}) \; . \tag{2.24}$$

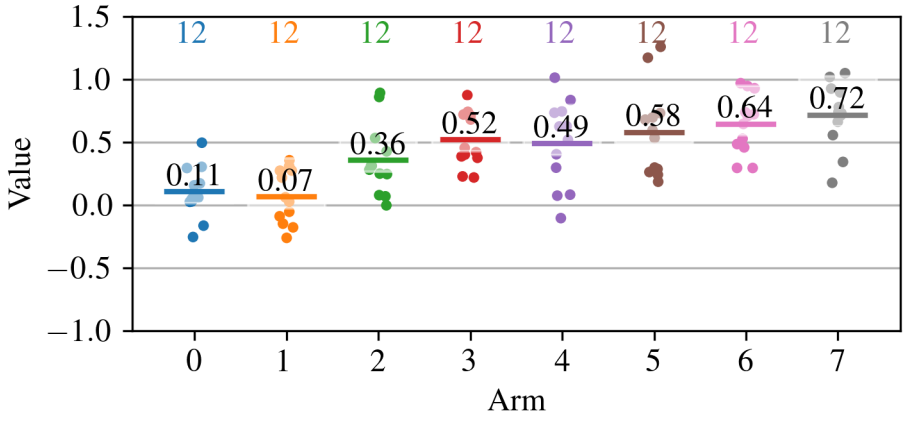
There are two common problem settings. In the fixed budget setting, the random variable $\mathrm{F}(\mathbf{a})$ can be sampled only a limited number of times, called the *computational budget* $N$. This means the algorithm can sample the function $N$ times

in total, and afterwards needs to predict the minimum $\mathbf{a}^*$. Algorithms for the fixed budget setting aim to maximize the probability of selecting the correct minimum. In the fixed confidence setting, a threshold on the probability of selecting the correct minimum is given, instead of a budget. There, the objective is to minimize the number of times F is sampled, to achieve a specified error probability.

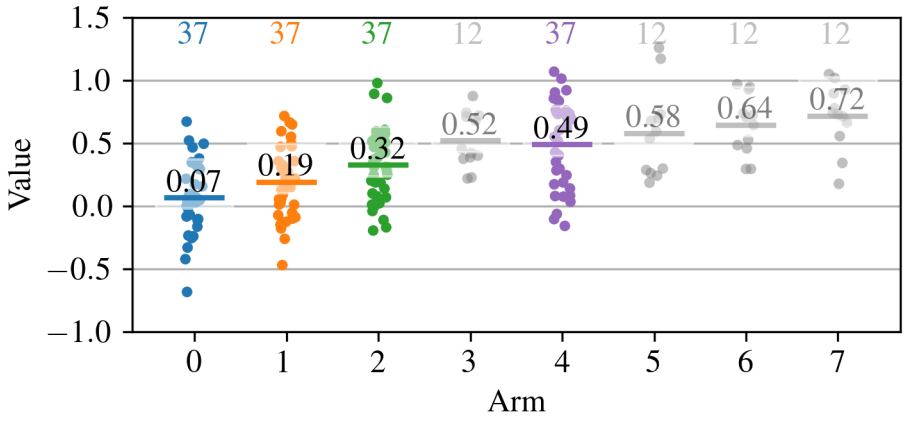
The fixed budget setting is of interest if the decision should be made in a fixed amount of time. One algorithm for this setting is the *sequential halving* algorithm, which searches for a minimum with a fixed computational budget $N$ [Karnin et al., 2013]. The idea of the sequential halving algorithm is to focus the computational budget in successive iterations on the most promising arms. In the initial iteration each arm is sampled with the same number of samples. This means multiple samples $\mathrm{F}(\mathbf{a})$ are observed for each $\mathbf{a} \in \mathbb{A}$. Then the worse half of the arms are removed. In the second iteration only the remaining arms are sampled. This continues until only one arm is remaining. As the set of arms is halved in each iteration, the algorithm requires $L = \lceil \log_2 |\mathbb{A}| \rceil$ iterations until a decision is made. The computational budget is spread evenly over the iterations, therefore in each iteration $\lfloor N/L \rfloor$ samples are taken. This means that in later iterations, when less arms are remaining, the remaining arms receive more samples per arm. Because in the first iteration every action needs to be sampled at least once, the algorithm requires a minimum computational budget of $\lceil |\mathbb{A}| \cdot \log_2 |\mathbb{A}| \rceil$.

Figure 2.1 shows an exemplary execution of the sequential halving algorithm. The MAB has eight arms and the arm with the lowest value is sought. The samples of each arm are normally distributed, with an expected value of $0.1, 0.2, ..., 0.8$ and a constant standard deviation of $\sigma = 0.3$. However, these values are unknown to the algorithm. The algorithm has a computational budget of $N = 300$, and requires $\log_2 8 = 3$ iterations. Therefore, 100 samples are available for each iteration.

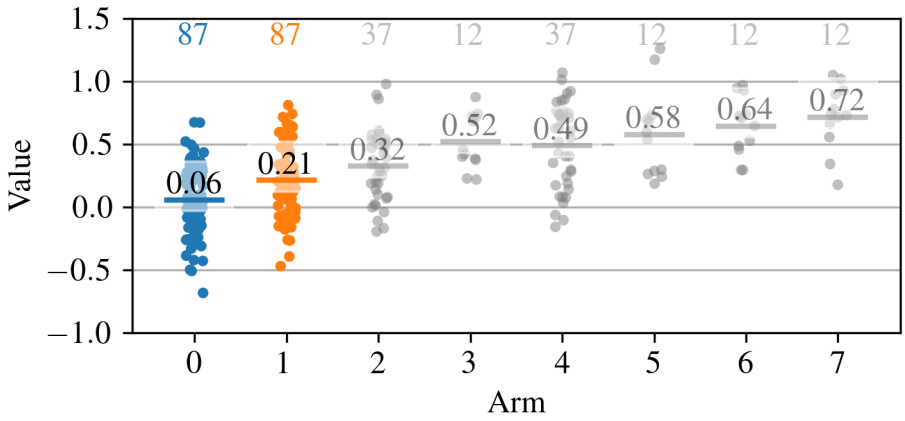
In the first iteration (Figure 2.1a), each arm is sampled $\lfloor 100/8 \rfloor = 12$ times. We can see that the true best arm has not the minimal sample mean, due to random fluctuations of F. Half of the arms (arms 3,5,6,7) with the highest values are assumed to be not the minimum and are rejected. These are not sampled any more in




(a) Iteration 0




(b) Iteration 1




(c) Iteration 2

**Figure 2.1:** An example of the sequential halving algorithm applied to a multi-armed bandit with eight arms. In three iterations, the algorithm successively focuses on the more promising arms. Each dot shows a sample of the value of an arm. The number above the dots shows the number of samples for the arm, and the horizontal line the average sample value.

the next iteration (Figure 2.1b). In this iteration, the remaining arms are sampled $100/4 = 25$ times, which means that the average is computed over 37 samples each. In the final iteration, the algorithm samples the arms 0 and 1 an additional 50 times and in the end commits to arm 0 as being the best, which is correct. It can be seen that the algorithm focuses the computational budget on the arms which are most promising. Due to rounding not the whole computational budget is used, but only 296 samples.

Being a stochastic algorithm, sequential halving is not guaranteed to find the correct solution. For the given problem it works well, finding the correct optimum in 9836 of 10 000 Monte Carlo runs. The second best arm was selected 164 times and the other arms were never selected.

## 2.2 Markov decision processes

*Markov decision processes* (MDPs) are a formal way to model sequential decision problems. Such problems consist of a sequence of decision steps. At each decision step, the current condition of the problem is given in form of a state variable. A decision consists of an action that changes the state in a probabilistic manner.

The defining property of an MDP is that the probability of transitioning from one state to a specific next state only depends on the current state and the current action. This property is called the *Markov property*. There are multiple reasons to assume the Markov property. One argument is that is makes modelling and algorithms easier. It is also a reasonable approximation for many practical problems. In [Powell, 2011, Ch. 5] the state is defined as the set of variables that contain all information to compute the state transition. With this definition a sequential decision process has by definition the Markov property. If the state transition also depends on earlier states, the state variable is not complete. In a similar way, so-called partially observable Markov decision processes (POMDPs) can be interpreted as MDPs as well, when defining the MDP using a sufficient statistic of the state. This will be discussed in Section 2.2.4.

### 2.2.1 Definition

In a Markov decision process, the current condition of the problem is represented as the *state* $\mathbf{x}_k \in \mathcal{X}$. The set of possible states is called *state space*. In this thesis we are interested in state spaces $\mathcal{X} \subseteq \mathbb{R}^P$, which means the $P$-dimensional real space.

The decision process advances in discrete steps and in each step an action can be selected to influence the transition from one state to another. We refer to these steps as *decision steps* $k \in \mathbb{N}$. If a decision is made in regular time intervals, we refer to the decision steps also as *time steps* and the time $t_k$ when the $k$-th decision is made is given by

$$t_k = k \cdot \Delta t \ . \tag{2.25}$$

Here $\Delta t$ is a fixed time interval. However, while such semantics are common, they are not necessary and the decision steps need not to correspond to any real time at all. Especially, in the algorithm discussed in Chapter 4 the decisions are not made in regular time intervals and the time at decision step $k$ depends on the previous state transitions.

At each decision step the MDP can be influenced by an *action* $\mathbf{a}_k \in \mathcal{A}$. The influence of the action on the state is modelled with a *transition function*

$$\mathrm{X}_{k+1} = f^a(\mathbf{x}_k, \mathbf{a}_k, \mathrm{W}_k^p) \ . \tag{2.26}$$

The superscript of $f^a$ should indicate that it models the influence of the action. The transition from $\mathbf{x}_k$ to $\mathrm{X}_{k+1}$ is called *state transition*. Note that the outcome of the state transition is a random variable. The randomness in the state transition is modelled by the *process noise* $\mathrm{W}_k^p$. We assume the process noise to be uncorrelated between different decision steps and its probability distribution to be known. After the action $\mathbf{a}_k$ and the random influence $\mathrm{W}_k^p$, the state transition is only dependent on the current state $\mathbf{x}_k$ and not on previous states $\mathbf{x}_i$ with $i < k$. This corresponds to the Markov property.

The above way to model the state transition is in the style used in optimal control theory, and is widely used in the literature [Bertsekas, 2017; Powell, 2011]. An

alternative and similarly common way to model the state transition is by specifying a *transition density*

$$p(\mathrm{X}_{k+1} \mid \mathbf{x}_k, \mathbf{a}_k) \tag{2.27}$$

which denotes the probability density of arriving in a state conditioned on the current state and the action. In the case of a finite state space, it is possible to represent (2.27) in form of a transition matrix that allows for effective mathematical manipulations. This formulation is commonly used in the mathematical theory of Markov decision processes [Hernández-Lerma and Lasserre, 1996; Puterman, 2005] as well as in the reinforcement learning literature [Sutton and Barto, 2018]. In this thesis the control theoretic notation is used.

The state transition incurs a *cost*, modelled by a *cost function* $c(\mathbf{x}_k, \mathbf{a}_k)$ or $c(\mathbf{x}_k, \mathbf{a}_k, \mathrm{W}_k^p)$. This cost models how desirable an action or state transition is. Instead of a cost, sometimes a *reward* is specified. This is the same concept, with the difference that a reward should be maximized, while a cost should be minimized.

The MDP starts in the *initial state* $\mathbf{x}_0$. Several assumptions on the length of an MDP can be made, resulting in different problem settings. For a finite number of decision steps, we arrive at a *finite horizon* MDP, defined as the tuple

$$\left(\mathcal{X}, \mathcal{A}, f^a, c, \mathbf{x}_0, p^{\mathrm{W}^p}, K\right) . \tag{2.28}$$

Here $K \in \mathbb{N}$ corresponds to the number of decision steps after which the process terminates. $p^{\mathrm{W}^p}$ refers to the probability density of the process noise $\mathrm{W}_k^p \sim p^{\mathrm{W}^p}$.

An MDP with infinite decision steps is called an *infinite horizon* MDP. In this case, the process is defined as the tuple

$$\left(\mathcal{X}, \mathcal{A}, f^a, c, \mathbf{x}_0, p^{\mathrm{W}^p}, \gamma\right) . \tag{2.29}$$

The *discount factor* $\gamma$ is used to weight the influence of future costs. A discount factor $0 < \gamma < 1$ is used to reduce the influence of costs that occur at later decision steps and therefore ensure that the total sum of costs remains finite. A discount factor of $\gamma = 1$ implies no discounting. This will also be discussed in Section 2.2.2.

In some applications, it is necessary to model an MDP that independent of the horizon terminates in some situations. In the context of this thesis, such a situation would be a successful localization. This behaviour can be modelled by including a special *termination state* $\mathcal{T}$, such that $f^a(\mathcal{T}, \mathbf{a}) = \mathcal{T}$ and $c(\mathcal{T}, \mathbf{a}) = 0$ for each action $\mathbf{a} \in \mathcal{A}$ [Bertsekas, 2017, Ch. 5]. Once the MDP reaches this state, it has effectively terminated as no further state changes are possible. In this thesis we define a set of termination states $\mathcal{T}^X \subset \mathcal{X}$. Once the process reaches one of these states, it terminates. Formally, this means that the MDP uses an augmented state space $\mathcal{X} \cup \{\mathcal{T}\}$, and for each $\mathbf{x} \in \mathcal{T}^X$ and each action $\mathbf{a} \in \mathcal{A}$ and process noise $\mathrm{W}^p$, the next transition is $f^a(\mathbf{x}, \mathbf{a}, \mathrm{W}^p) = \mathcal{T}$ with cost $c(\mathbf{x}, \mathbf{a}) = 0$.

If the cost function is nonnegative, the cost can be interpreted as a distance between the states. For an infinite horizon MDP with termination state, the minimization of the total cost has then the intuition of finding a shortest path from the initial state to a termination state. Therefore, this case is called a *stochastic shortest path problem* [Bertsekas, 2017, Ch.5]. A discount factor of $\gamma = 1$ is often used and the problem is formally stated as the tuple

$$\left(\mathcal{X}, \mathcal{A}, f^a, c, \mathbf{x}_0, p^{\mathrm{W}^p}, \mathcal{T}^X\right) . \tag{2.30}$$

We define the random variable K as the index of the first decision step $k$ when $\mathbf{x}_k \in \mathcal{T}^X$, such that the last meaningful action was performed at decision step $\mathrm{K}-1$. This can be seen as the random analogue to $K$ in a finite horizon MDP.

### 2.2.2 Optimal policies and Bellman's equation

The MDP itself does not specify a method to select the actions. When this is done in an automated way, the decision algorithm is called a *policy*. Generally, a policy

$$\pi_k : \mathcal{X} \to \mathcal{A} \tag{2.31}$$

is a mapping from the state space to the action space. Exemplary implementations are

- a lookup table, which is especially common for discrete state spaces
- an algorithm that solves an optimization problem, for example in model predictive control
- a parameterized function, commonly used in reinforcement learning
- or simply a random selection of an action.

In finite horizon MDPs, the policy $\pi_k$ often changes with the decision step. If a policy

$$\pi : \mathcal{X} \to \mathcal{A} \tag{2.32}$$

is the same for each decision step, it is called a *stationary policy* [Bertsekas, 2017, Ch. 5]. This is often the case in infinite horizon MDPs. In the remainder of this section we will focus on stationary policies and infinite horizon MDPs.

With a given policy, the Markov decision process becomes a *Markov process*, which means a sequence of random state transitions with transition function

$$\mathrm{X}_{k+1} = f^{\pi}(\mathbf{x}_k, \mathrm{W}_k^p) = f^a(\mathbf{x}_k, \pi(\mathbf{x}_k), \mathrm{W}_k^p) \; . \tag{2.33}$$

Based on this sequence, the *value function* $v^{\pi} : \mathcal{X} \to \mathbb{R}$ can be defined. This function

$$v^{\pi}(\mathbf{x}) = \mathbb{E}\left[ \sum_{i=0}^{\infty} \gamma^i c(\mathrm{X}_i, \pi(\mathrm{X}_i)) \;\middle|\; \mathrm{X}_0 = \mathbf{x} \right] \tag{2.34}$$

is defined as the expected *future costs* when starting in a state $\mathbf{x}$ and following a policy $\pi$. Another name used in the literature for this function is the *cost-to-go function* [Bertsekas, 2017, Ch. 1]. The role of a discount factor $\gamma \in \mathbb{R}, 0 < \gamma < 1$ is to ensure that $v^{\pi}$ remains finite. With a discount factor of $\gamma = 1$, $v^{\pi}$ is finite if the policy eventually reaches a termination state, as defined in the stochastic shortest path problem. We also refer to $v^{\pi}(\mathbf{x})$ as the *value* of $\mathbf{x}$.

In a similar way, the *action value*

$$q^{\pi}(\mathbf{x}, \mathbf{a}) = \mathbb{E}\left[ c(\mathbf{x}, \mathbf{a}) + \gamma \cdot v^{\pi}(f^a(\mathbf{x}, \mathbf{a}, \mathrm{W}^p)) \right] \tag{2.35}$$

also called *Q-value*, can be defined. This corresponds to the future cost encountered when taking an action $\mathbf{a}$ in a state $\mathbf{x}$ and following the policy $\pi$ afterwards.

It is reasonable to judge a policy based on the total cost encountered during the execution of the MDP. This corresponds to the value of $\mathbf{x}_0$. The optimization problem of an MDP is therefore

$$\pi^* = \underset{\pi}{\operatorname{argmin}}\, v^\pi(\mathbf{x}_0) \tag{2.36}$$

which means to find a policy that minimizes the expected costs of the initial state [Bertsekas, 2017, Ch. 1]. If such a policy exists, $\pi^*$ is called the *optimal policy*. The corresponding value function $v^{\pi^*}$ of $\pi^*$ is called the *optimal value function* and denoted by $v^*$. Similarly, we denote $q^{\pi^*}$ as $q^*$. The optimal value function can be characterized by the recursive equation

$$v^*(\mathbf{x}) = \min_{\mathbf{a}\in\mathcal{A}} \mathbb{E}\left[c(\mathbf{x},\mathbf{a}) + \gamma \cdot v^*(f^a(\mathbf{x},\mathbf{a},\mathrm{W}^p))\right] \tag{2.37}$$

$$= \min_{\mathbf{a}\in\mathcal{A}} q^*(\mathbf{x},\mathbf{a})\ . \tag{2.38}$$

Equation (2.37) is known as *Bellman's equation* or *Bellman's optimality equation* [Bertsekas, 2017, Ch. 5], [Powell, 2019]. This equation captures the intuition that an optimal policy consists of a sequence of optimal decisions, where the sum of the *immediate cost* $c(\mathbf{x}_k,\mathbf{a})$ and the discounted future cost $\gamma \cdot v^*(\mathbf{x}_{k+1})$ should be minimized. The form of equation (2.38) captures the idea that in a state $\mathbf{x}_k$, the policy should select the action with the smallest action value.

In many problems, computing the optimal policy is not feasible and approximate solution methods need to be used. The next section gives an overview about such solution methods. The value $v^\pi(\mathbf{x}_0)$ provides a way to compare the performance of different policies on a given MDP and therefore to judge whether one policy or another is better. To compute the performance of a policy, equation (2.34) can be evaluated with multiple Monte Carlo runs, to estimate the value $v^\pi(\mathbf{x}_0)$.

### 2.2.3 Solution methods

Approximate dynamic programming algorithms can be divided into *online* and *offline* algorithms. An online algorithm solves an optimization problem during the execution of the MDP. This allows the algorithm to focus on the current state $\mathbf{x}_k$ of the MDP without the requirement to compute an optimal action for each possible state. In comparison, an offline algorithm computes a policy once which can then be used each time the MDP is executed.

#### Backward dynamic programming

In a finite horizon MDP with finite state space $\mathbb{X}$, finite action space $\mathbb{A}$, and $\mathrm{W}^p$ taking values from a finite set of possible process noises $\mathbb{W}^p$, the optimal value function can be solved by going backwards from the last decision step $K-1$ to the first. By initializing $v_K^*(\mathbf{x}) = 0$ for each $\mathbf{x} \in \mathbb{X}$, the discrete undiscounted finite horizon form of Bellman's equation (2.37)

$$v_k^*(\mathbf{x}) = \min_{\mathbf{a}\in\mathbb{A}} \mathbb{E}\left[c(\mathbf{x},\mathbf{a}) + v_{k+1}^*(f^a(\mathbf{x},\mathbf{a},\mathrm{W}_k^p))\right] \tag{2.39}$$

$$= \min_{\mathbf{a}\in\mathbb{A}} c(\mathbf{x},\mathbf{a}) + \sum_{\mathbf{w}^p\in\mathbb{W}^p} \mathcal{P}(\mathrm{W}^p = \mathbf{w}^p) \cdot v_{k+1}^*(f^a(\mathbf{x},\mathbf{a},\mathbf{w}^p)) \tag{2.40}$$

$$= \min_{\mathbf{a}\in\mathbb{A}} c(\mathbf{x},\mathbf{a}) + \sum_{\hat{\mathbf{x}}\in\mathbb{X}} \mathcal{P}(\mathrm{X}_{k+1} = \hat{\mathbf{x}} \mid \mathbf{x},\mathbf{a}) \cdot v_{k+1}^*(\hat{\mathbf{x}}) \tag{2.41}$$

gives the value for every $v_{K-1}^*(\mathbf{x})$. This can be repeated until every $v_0^*(\mathbf{x})$ is known. The action $\mathbf{a}_{k\mathbf{x}}^*$ that achieves the minimum in (2.39) can be stored as the action that minimizes the total expected costs in state $\mathbf{x}$. This is the action a policy should execute and we set

$$\pi_k(\mathbf{x}) = \mathbf{a}_{k\mathbf{x}}^* \,. \tag{2.42}$$

This algorithm is known as *backward dynamic programming* [Powell, 2011, Ch. 3] and requires $\mathcal{O}(K \cdot |\mathbb{X}| \cdot |\mathbb{W}^p| \cdot |\mathbb{A}|)$ operations in formulation (2.40) and $\mathcal{O}(K \cdot |\mathbb{X}^2| \cdot |\mathbb{A}|)$ in formulation (2.41). The policy itself is given by a lookup table with $K \cdot |\mathbb{X}|$ entries. Note that the policy is nonstationary.

This algorithm illustrates an important concept, known as *curse of dimensionality*. We first note that a computation linear in the size of the state space, action

space, and the set of possible process noises, does sound feasible. However, a problem appears if the state space is multidimensional as in

$$\mathbb{X} = \mathbb{X}_1 \times \mathbb{X}_2 \times ... \mathbb{X}_P \ . \tag{2.43}$$

As example, we can consider the discretized state space of an airplane consisting of its position and its velocity. We assume that each dimension is discretized into only 10 values, which is extremely small. Then the combined state space with a 3-dimensional position and 3-dimensional velocity has a size of $10^6 = 1,000,000$. The same problem appears with the set of process noises and the action space. This is a major motivation of online algorithms that can focus the computation on the current state and do not need to consider the whole state space.

## Myopic policies

A *myopic policy* is an online algorithm that minimizes the immediate cost [Powell, 2011, Ch. 6]. To perform this minimization it only considers the current action and no future actions. Another commonly used name for such a policy is *greedy policy*. For a deterministic cost function the policy is given by

$$\pi(\mathbf{x}_k) = \underset{\mathbf{a} \in \mathcal{A}}{\operatorname{argmin}} \ c(\mathbf{x}_k, \mathbf{a}) \tag{2.44}$$

and for a stochastic cost function the expected cost is minimized by

$$\pi(\mathbf{x}_k) = \underset{\mathbf{a} \in \mathcal{A}}{\operatorname{argmin}} \ \mathbb{E}\left[c(\mathbf{x}_k, \mathbf{a}, \mathrm{W}_k^p)\right] \ . \tag{2.45}$$

In some cases, the cost function depends only on the state and not on the action, that means the cost function $c(\mathbf{x}, \mathbf{a})$ can also be written as $c(\mathbf{x})$. Then a single state transition needs to be made, leading to the policy

$$\pi(\mathbf{x}_k) = \underset{\mathbf{a} \in \mathcal{A}}{\operatorname{argmin}} \ \mathbb{E}\left[c(f^a(\mathbf{x}_k, \mathbf{a}, \mathrm{W}_k^p))\right] \ . \tag{2.46}$$

A common variation is to use a cost function that is different to the original MDP. In this case, the substitute $c^S(\mathbf{x}, \mathbf{a})$ is called a *surrogate cost function*. Myopic policies have the advantage of small computational requirements as only the direct state transitions from the current state are required. They can be used for finite and infinite horizon MDPs. They have the disadvantage that future costs are not considered and the selected action might lead to higher costs in future decision steps.

## Deterministic lookahead

*Deterministic lookahead* or *model predictive control* (MPC) is an online algorithm that solves at each decision step a deterministic control problem over a fixed horizon [Powell, 2019]. It uses a deterministic approximation $f^d$ of the transition function $f^a$, to model the state transition as

$$\mathbf{x}_{k+1} = f^d(\mathbf{x}_k, \mathbf{a}) \ . \tag{2.47}$$

It is possible to use the original transition function with either a nominal noise or zero noise and set $f^d(\mathbf{x}, \mathbf{a}) = f^a(\mathbf{x}, \mathbf{a}, \mathbf{0})$. At each decision step $k$, the deterministic lookahead algorithm solves the optimization problem

$$\mathbf{a}^*_{k[1:H]} = \underset{\mathbf{a}_{1:H} \in \mathcal{A}^H}{\operatorname{argmin}} \sum_{i=1}^{H} \gamma^{i-1} c(\mathbf{x}^f_i, \mathbf{a}_i) \ , \tag{2.48}$$

where the sequence of future states $\mathbf{x}^f_i$ starts with the current state, that means $\mathbf{x}^f_1 = \mathbf{x}_k$, and all future states are the result of the deterministic state transition (2.47). The number $H$ of considered state transitions is called the *planning horizon* of the algorithm. After computing $\mathbf{a}^*_{k[1:H]}$, this algorithm implements only the first action of this sequence. This means the policy is given by

$$\pi(\mathbf{x}_k) = \mathbf{a}^*_{k1} \ . \tag{2.49}$$

After this action is executed, the MDP performs a state transition and the above process is repeated. In this way, the planning horizon considers always the next $H$ decision steps and is therefore also called a *receding horizon*.

Together with differentiable transition and cost functions, a local minimum of (2.48) can be computed using the algorithms discussed in Section 2.1.1. Other solution techniques are tree search for a discrete action space, or specialized methods as for example differentiable dynamic programming [Tassa et al., 2012, 2014].

As mentioned above, the algorithm of deterministic lookahead is also called *model predictive control*. A more stricter interpretation of MPC would refer to a set of deterministic lookahead algorithms whose cost function is based on the difference between the true state and a desired state. This is a common problem in an industrial setting, for example if the temperature in a boiler of a chemical plant must follow a nominal temperature sequence as close as possible. A restriction to such a cost function allows for specialized algorithms [Camacho and Bordons, 2007]. A more general interpretation of MPC would be any algorithm with a receding horizon. This can be defined as an algorithm that creates a finite horizon MDP and computes for it a policy $\hat{\pi}_k$ [Powell, 2019]. This policy might be a sequence of actions as above, but also any more complex policy, for example resulting from backwards dynamic programming. Again, this policy is only executed for the current state

$$\pi(\mathbf{x}_k) = \hat{\pi}_k(\mathbf{x}_k) \tag{2.50}$$

and after the state transition the whole algorithm is executed again for the next state. To avoid confusion about the term *model predictive control*, the term *deterministic lookahead* is used in this thesis.

## Policy iteration

The technique of *policy iteration* creates a sequence $\pi_1, \pi_2, \pi_3, \ldots$ of continually improving stationary policies. It is an offline algorithm that computes the policy sequence once, resulting in a policy that can then be used without further computation.

The algorithm starts with an initial policy $\pi_1$. In each iteration $l$ the value function of the policy $\pi_l$, $v_l^\pi(\mathbf{x})$ is computed for each state $\mathbf{x} \in \mathcal{X}$. This step is called *policy evaluation*. In the *policy improvement* step a new policy

$$\pi_{l+1}(\mathbf{x}) = \underset{\mathbf{a}\in\mathcal{A}}{\operatorname{argmin}} \, \mathbb{E}\left[c(\mathbf{x},\mathbf{a}) + \gamma \cdot v_l^\pi(f^a(\mathbf{x},\mathbf{a},\mathrm{W}^p))\right] \tag{2.51}$$

is computed for each state. Policy iteration can be used for infinite horizon problems with a finite state space. In this case the policy evaluation can be done efficiently based on a steady state analysis of the Markov chain indicated by $\pi_l$ [Powell, 2011, Ch. 3]. Also for a finite state space, $\pi_l$ can be efficiently stored as a lookup table. The sequence of policies continually improves in the sense that the value function improves with $v_{l+1}^\pi(\mathbf{x}) \leq v_l^\pi(\mathbf{x})$ for each state $\mathbf{x}$ [Bertsekas, 2017, Ch. 5]. As policy iteration considers the whole state space in each iteration, it also suffers under the curse of dimensionality.

## Policy rollout

The *policy rollout* algorithm is an online algorithm that uses an existing policy, the so-called *base policy* to approximate future actions [Bertsekas and Castañón, 1999]. It can be interpreted as an online version of policy iteration, focused on the current state. We introduce policy rollout for an infinite horizon MDP, but it can be similarly used for finite horizon MDPs.

The base policy

$$\pi^B : \mathcal{X} \to \mathcal{A} \tag{2.52}$$

is an existing policy for the problem. For instance, this could be a myopic policy, a problem specific heuristic, or any other policy. Using (2.34) and (2.35), we can define the action value of $\pi^B$

$$q^B(\mathbf{x},\mathbf{a}) = \mathbb{E}\left[c(\mathbf{x},\mathbf{a}) + \sum_{i=1}^{\infty} \gamma^i c(\mathrm{X}_i, \pi^B(\mathrm{X}_i)) \;\middle|\; \mathrm{X}_0 = \mathbf{x}\right] . \tag{2.53}$$

The notation $q^B$ is a short form for $q^{\pi^B}$. The future states $\mathrm{X}_{i+1} = f^a(\mathrm{X}_i, \pi^B(\mathrm{X}_i), \mathrm{W}_i^p)$, $i \geq 1$, are based on the random process noise $\mathrm{W}_i^p$ and the decisions of the base pol-

icy $\pi^B$. The immediate next state $\mathrm{X}_1 = f^a(\mathbf{x}, \mathrm{a}, \mathrm{W}_0^p)$ is based on the action $\mathbf{a}$ instead of the base policy. Using $q^B$, the *rollout policy* is defined as

$$\pi^R(\mathbf{x}) = \underset{\mathbf{a}\in\mathcal{A}}{\operatorname{argmin}}\, q^B(\mathbf{x}, \mathbf{a}) \; . \tag{2.54}$$

The action value of the base policy (2.53) can be interpreted in the following way: $q^B$ is the expected future discounted cost when performing action $\mathbf{a}$ in state $\mathbf{x}$ and afterwards using the base policy to determine each future action. This step can be interpreted as the policy evaluation of $\pi^B$. The rollout policy then selects the action that has the smallest expected discounted future costs, when following the base policy for future actions. This is similar to the policy improvement step. Together, policy rollout can be considered as a single step of policy iteration, focused on the current state and performed online.

An important concept for the policy rollout algorithm is the idea of a *sequentially consistent* policy [Bertsekas et al., 1997; Secomandi, 2003]. Sequential consistency means intuitively that a policy does not produce different actions as the MDP progresses. Let $\mathbf{w}_k^p, \mathbf{w}_{k+1}^p, \mathbf{w}_{k+2}^p, \ldots$ be a sequence of process noise realizations and $\mathbf{x}_k$ the current state. Then the base policy would create a sequence

$$\mathbf{x}_k, \mathbf{a}_k, \mathbf{x}_{k+1}, \mathbf{a}_{k+1}, \mathbf{x}_{k+2}, \ldots \tag{2.55}$$

where the future states are deterministic due to the fixed process noise sequence. The base policy is called sequentially consistent, if after an execution of action $\mathbf{a}_k$ and arrival in state $\mathbf{x}_{k+1}$, it would generate a future sequence

$$\mathbf{x}_{k+1}, \mathbf{a}_{k+1}, \mathbf{x}_{k+2}, \ldots \; . \tag{2.56}$$

This means that the same actions are made for the same states in the same decision steps. An important category of sequentially consistent policies are deterministic stationary policies. An example of a base policy that would not be sequentially consistent would be a policy that returns always the same sequence of actions $\mathbf{a}_1, \mathbf{a}_2, \ldots$,

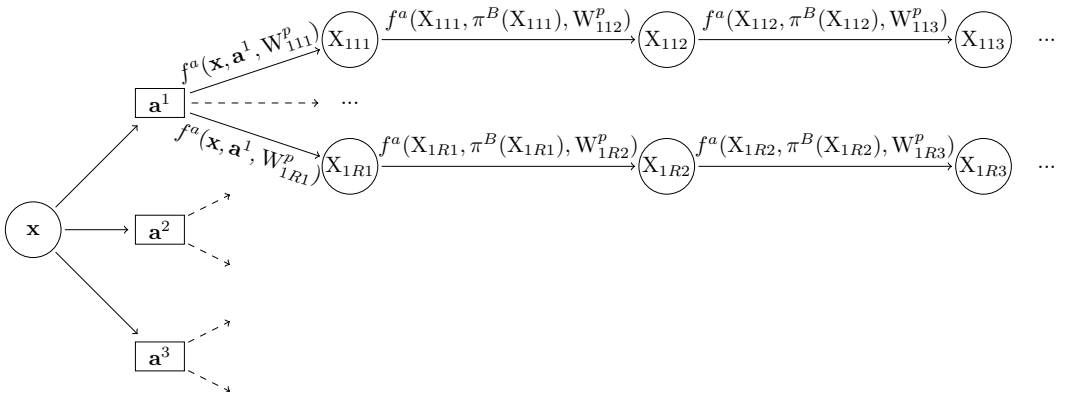



**Figure 2.2:** Policy rollout using Monte Carlo sampling.

during the rollout. If action $\mathbf{a}_1$ were to be implemented, it would return the same sequence $\mathbf{a}_1, \mathbf{a}_2, ...$ in the next decision step instead of starting with $\mathbf{a}_2$. This would lead to a different assessment of the action values. Note that in this thesis, the definition of the base policy (2.52), assumes a stationary policy which is therefore also sequentially consistent.

If the base policy is sequentially consistent, the resulting rollout policy is performing equal or better than the base policy, which means that

$$v^R(\mathbf{x}) \leq v^B(\mathbf{x}) \quad \text{for each } \mathbf{x} \in \mathcal{X} \ . \tag{2.57}$$

Similar to $q^B$, $v^B$ is a short form for $v^{\pi^B}$ and $v^R$ for $v^{\pi^R}$. This property is the case because every decision of the rollout policy cannot be worse than a decision taken the base policy. Equation (2.57) is known as *cost improvement property* [Bertsekas, 2005], *rollout improvement property* [Secomandi, 2003; Goodson et al., 2013] or *policy improvement property* [Chong et al., 2009]. In this thesis the term rollout improvement property is used.

A common way to compute $q^B$ is via Monte Carlo sampling [Tesauro and Galperin, 1996]. To compute (2.53), the infinite sum can be replaced by a partial sum $\sum_{i=1}^{H}$ for a planning horizon of $H$. This is referred to as a *truncated rollout*. Note, that with a finite number of Monte Carlo samples or a truncated rollout, the rollout algorithm does not compute the true action value $q^B$ and the rollout improvement property is not guaranteed to hold.

It is common that a finite action space $\mathbb{A}$ is used in (2.54). Then, the rollout policy can be computed by performing $R$ Monte Carlo samples for each action $\mathbf{a} \in \mathbb{A}$ and select that action with the smallest sampled cost. Figure 2.2 visualizes the policy rollout algorithm using Monte Carlo sampling. The action space $\mathbb{A} = \{\mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3\}$ consists of three actions. All variables use an index $aji$ where $a$ corresponds to the action, $j$ to the rollout number and $i$ for the future decision step. For each action $R$ rollouts are performed. After execution of the initial action, different realizations of the process noises $\mathrm{W}^p_{111}, ..., \mathrm{W}^p_{1R1}$ lead to $R$ potentially different states. Each future action is decided by the base policy and future state transition depends on the sampled process noise. A major feature of the policy rollout is that its computation effort is linear with respect to the planning horizon. This is the case because, contrary to for example a tree search, future actions are not explicitly optimized but are instead determined by the base policy.

In the case of a stochastic shortest path problem, we state the action value of the base policy as

$$q^B(\mathbf{x}, \mathbf{a}) = \mathbb{E}\left[ c(\mathbf{x}, \mathbf{a}) + \sum_{i=1}^{\mathrm{K}-1} c(\mathrm{X}_i, \pi^B(\mathrm{X}_i)) \;\middle|\; \mathrm{X}_0 = \mathbf{x} \right] . \tag{2.58}$$

The major difference to (2.53) is that the costs are undiscounted and only summed up until the first termination state is reached. Instead until infinity, the sum runs until $\mathrm{K} - 1$, where the random variable K denotes the first future decision step $i$ when $\mathrm{X}_i \in \mathcal{T}^X$, that means $\mathrm{X}_i$ is a termination state. Effectively, this means that each individual rollout is performed until the MDP terminates.

## Other solutions

There is a multitude of algorithms to solve Markov decision processes and only a brief overview relevant to this thesis was given in the above paragraphs. This section mentions some other algorithms to solve MDPs.

*Value iteration* is a technique that computes a sequence of value functions $v_l(\mathbf{x})$ on basis of Bellman's equation. It is similar to backwards dynamic programming, but does not require the MDP to have a finite horizon. Under suitable assump-

tions, including a finite state and action space, it can be proven that this sequence converges to the optimal value function $v^*$ [Bertsekas, 2017, Ch. 5].

*Policy gradient methods* use a policy $\pi_\phi$, parameterized by a vector $\phi$ that computes the probabilities of different actions and is differentiable with regard to $\phi$. Such a policy could, for example, be a neural network. The differentiability of the policy is used to update the parameter vector, making actions with empirically lower future costs more likely [Sutton and Barto, 2018, Ch. 13].

*Q-learning* and *SARSA* both aim to learn an action value function $q_\phi(\mathbf{x}, \mathbf{a})$ for each pair of state and action. This function is similarly parameterized by a vector $\phi$, which is updated based on empirical costs that are received when executing the policy [Sutton and Barto, 2018, Ch. 6].

Overviews over these and other algorithms can be found in [Sutton and Barto, 2018; Bertsekas, 2012, 2017; Powell, 2011, 2019; Thrun et al., 2005].

### 2.2.4 Partial observability

The previous discussion assumed that at each decision step $k$, the state $\mathbf{x}_k$ is observable. In many problems of interest however, the full state is not directly available to the policy. Instead, a *measurement* $\mathbf{z}_k$ is generated at each decision step $k$, which is corrupted by *measurement noise*. In this case, the problem is called a *partially observable Markov decision process* (POMDP). The measurement is modelled with a *measurement function* $h$ and

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{w}_k^m \,. \tag{2.59}$$

Here $\mathbf{w}_k^m$ represents additive measurement noise. The set $\mathcal{Z} \subseteq \mathbb{R}^Z$ of possible measurements is called the *measurement space*. We also make the common modelling assumption that $\mathbf{w}_k^m$ is the realization of a multivariate normal distributed random variable with known covariance matrix $\mathbf{R}_k$ and an expected value of zero.

This measurement model with a measurement noise $\mathbf{w}_k^m$ is in the style commonly used in sensor data fusion, discussed in Chapter 3. Similar to the transition function of MDPs, an alternative and also common formulation would be a proba-

bility distribution $p(\mathbf{z}_k \mid \mathbf{x}_k)$ of the measurement. In the literature, this probability distribution also often depends on the action [Kaelbling et al., 1998; Shani et al., 2012]. Here we assume that the measurement is only dependent on the current state. We will later recover the dependency on the action by augmenting the state with a *sensor state* that influences the measurement. This is a natural way to model the problem if an action changes the measurements permanently, for example by moving a platform.

At decision step $k$, the *information vector*

$$\mathbf{i}_k = (\mathbf{z}_0, \mathbf{a}_0, \mathbf{z}_1, ..., \mathbf{a}_{k-1}, \mathbf{z}_k) \tag{2.60}$$

contains all information available to the policy [Bertsekas, 2017, Ch. 4]. However, such a vector is of increasing size with each decision step, and therefore often impractical. Due to the Markov property, only state $\mathbf{x}_k$ and action $\mathbf{a}_k$ influence the next state $\mathbf{x}_{k+1}$. It is therefore sufficient to summarize the information vector into a belief about the current state $\mathbf{x}_k$. This belief

$$b_k := p(\mathrm{X}_k \mid \mathbf{i}_k) \tag{2.61}$$

$$= p(\mathrm{X}_k \mid \mathbf{z}_{0:k}, \mathbf{a}_{0:k-1}) \tag{2.62}$$

is a probability distribution about the current state, conditioned on the previous measurements and actions. The current state is modelled as a random variable $\mathrm{X}_k$ because its true value $\mathbf{x}_k$ is not known to the policy. Section 3.1 describes in more detail how such a belief can be created. If the belief summarizes all available information about the current state, it is called a *sufficient statistic* [Bertsekas, 2017, Ch. 4]. A special case appears if parts of the state space are directly observable. For example if the state

$$\mathbf{x}_k = \begin{pmatrix} \mathbf{x}_k^u \\ \mathbf{x}_k^o \end{pmatrix} \tag{2.63}$$

consists of two parts, where $\mathbf{x}_k^u$ is unobservable and only available by noisy measurements and $\mathbf{x}_k^o$ is directly observable. We keep the notation of the belief being a

probability distribution over the state space and represent the probability distribution of $\mathbf{x}_k^o$ by the *Dirac delta function* $\delta$ as

$$p(\mathrm{X}_k = (\mathbf{x}^u, \mathbf{x}^o) \mid \mathbf{i}_k) = p(\mathrm{X}_k^u = \mathbf{x}^u \mid \mathbf{i}_k) \cdot \delta(\mathbf{x}^o - \mathbf{x}_k^o) \ . \tag{2.64}$$

This should be understood only as a notation to keep the idea that the belief is a probability distribution over the state space. In a practical implementation, $\mathbf{x}_k^o$ is simply stored as vector.

The policy in a POMDP is defined as a function

$$\pi : \mathcal{B} \to \mathcal{A} \tag{2.65}$$

from the current belief to an action. Here, $\mathcal{B}$ refers to the set of all possible beliefs, also called *belief space*. Similar to MDPs, a cost function $c(b_k, \mathbf{a})$ is defined. It is possible to define this as the expected value of a cost function defined on the state

$$c(b_k, \mathbf{a}) = \mathbb{E}\left[c(\mathrm{X}_k, \mathbf{a})\right] \tag{2.66}$$

where $\mathrm{X}_k$ is distributed according to $b_k$. However, in Section 3.3 we will encounter the field of sensor management, where the goal is to improve on the belief about the state. Here, the cost function is often defined directly on the belief, typically representing its information content, for example

$$c(b_k, \mathbf{a}) = \mathcal{H}^D(b_k) \tag{2.67}$$

where $\mathcal{H}^D$ is the differential entropy, see Section 3.1.4. Using the notation above, we can now specify an infinite horizon POMDP as a tuple

$$(\mathcal{X}, \mathcal{Z}, \mathcal{B}, f^a, h, c, b_0, p^{\mathrm{W}^p}, p^{\mathrm{W}^m}, \gamma) \tag{2.68}$$

of the state space $\mathcal{X}$, transition function $f^a$, process noise distribution $p^{\mathrm{W}^p}$, and discount factor $\gamma$ of an underlying MDP, as well as the measurement space $\mathcal{Z}$, belief

space $\mathcal{B}$, measurement function $h$, cost function $c$, distribution of the measurement noise $p^{\mathrm{W}^m}$, and initial belief $b_0$ specific to the POMDP. Similar to MDPs, we also define a POMDP version of the stochastic shortest path problem as tuple

$$(\mathcal{X}, \mathcal{Z}, \mathcal{B}, f^a, h, c, b_0, p^{\mathrm{W}^p}, p^{\mathrm{W}^m}, \mathcal{T}^B) \, . \tag{2.69}$$

In a similar way to the MDP version, we assume a discount factor of $\gamma = 1$ and a set of *termination beliefs* $\mathcal{T}^B$. The algorithm presented in Chapter 4 is based on this problem formulation.

From the view of the policy, a POMDP now looks as described in the following: At decision step $k$, the policy has knowledge about the belief $b_k$ and selects an action $\mathbf{a}_k$. This results in a cost $c(b_k, \mathbf{a}_k)$, an observation $\mathbf{z}_{k+1}$ and with this observation in a new belief state $b_{k+1}$. It can be clearly seen that this mirrors the sequential decision procedure of an MDP, only with a belief instead of a state. Thus, a POMDP is an MDP on the belief space. This allows us to apply the previous solution techniques also in the partially observable case.

A challenge with POMDPs is that the belief space is way larger than the original state space. As an example, let us consider a finite state space $\mathbb{X}_2 = \{1, 2\}$ with only two possible states. A belief would specify the probability

$$b_k = p(\mathrm{X}_k = 1) = 1 - p(\mathrm{X}_k = 2) \, . \tag{2.70}$$

The corresponding belief space $\mathcal{B}_2 = [0, 1] \subset \mathbb{R}$ would be an interval of the real values containing an uncountable number of possible beliefs. If the finite state space $\mathbb{X}_{10} = \{1, 2, ...10\}$ would consist of 10 states, the belief space would be the set $\mathcal{B}_{10} = \{\mathbf{b} \in [0, 1]^9 : \|\mathbf{b}\|_1 = 1\}$ of all 9-dimensional vectors with a sum of 1. If the state space is continuous, the belief space is similarly larger, even if restricted to a specific type of belief representations like multivariate Gaussians. As example, for a 2-dimensional state space $\mathcal{X}_2 = \mathbb{R}^2$, the belief space $\mathcal{B} \subset \mathbb{R}^5$ of Gaussians would be 5-dimensional. It would consist of all possible means of the Gaussians, as well as all possible upper triangular parts of a $2x2$ covariance matrix.

Using an offline algorithm like policy iteration or value iteration in a POMDP is therefore much more difficult than in an MDP. Offline algorithms can use the fact that under some conditions the optimal value function $v_k^*(b)$ is piecewise linear convex. One of the requirements is that the state space and the horizon is finite. The value function can then be represented using a finite number of linear functions, allowing the solution by exact backwards dynamic programming [Smallwood and Sondik, 1973]. While this exact computation is only feasible for small state spaces, advancements have been made with the arrival of *point-based value iteration*. These algorithms compute the linear functions representing the value function only for a limited subset of beliefs [Shani et al., 2012]. However, while these algorithms generally support larger state spaces, they still need to be finite and have sizes that can easily be surpassed when discretizing continuous state spaces.

Due to the difficulty with offline algorithms, this thesis uses an online algorithm for planning. In particular, it uses the policy rollout algorithm described in Section 2.2.3. This algorithm takes the uncertainty of the problem into account and works in a non-discrete state space with non-discrete measurements. Major challenges in using it are the definition of a good base policy and the required computational effort due to Monte Carlo sampling.

# Chapter 3

# Sensor Data Fusion and Sensor Management

In this thesis we are interested in localizing an emitter of RF signals. We obtain information about its position using a *sensor*, which is a device that converts the *target state* $\mathbf{x}^t \in \mathcal{X}^t \subseteq \mathbb{R}^T$ of an object into a measurement $\mathbf{z} \in \mathcal{Z} \subseteq \mathbb{R}^Z$. An example for a target state would be the position of the target. An example for a measurement would be the direction from the sensor towards the target. Figure 3 visualizes this process.

$$\mathbf{x}^t \longrightarrow \boxed{\text{sensor}} \longrightarrow \mathbf{z} \ .$$

**Figure 3.1:** A sensor transforms the true target state into a measurement.

The sensor might have several parameters, which affect this measurement process. These could be the sensor position and orientation, or its configuration. We summarize these values in the *sensor state* $\mathbf{x}^s \in \mathcal{X}^s \subseteq \mathbb{R}^S$. As is common in the literature we will call the measured object the *target*. We denote $\mathcal{X}^t$ as the *target state space* and $\mathcal{Z}$ as the *measurement space*. The discussion in this chapter focuses on the case of a single target.

Sensors commonly produce multiple measurements over time, which individually contain only limited information about the target state. The field of *sensor data fusion* is concerned with combining those measurements into a single belief about the current situation. This belief is a probability distribution over the target

state, which is updated based on Bayes' theorem. In Section 3.1 we will discuss the Bayes filter algorithm, different options to represent the belief, as well as metrics which allow us to quantify their accurateness.

Direction finding (DF) sensors measure the direction or *bearing* of an incoming signal and are discussed in Section 3.2. The Fisher information is used to quantify the information a bearing measurement gives about the target position. We will see that this information is highly dependent on the sensor-to-target geometry, and derive optimal geometries given the true target position.

However, a mobile sensor system cannot directly use these optimal sensor-to-target geometries. First, the true target position is normally unknown and only an uncertain belief about it is available. Therefore, the decision on where to move the sensor system is made under uncertainty. Second, the optimal sensor-to-target geometry might not necessarily be the optimal next measurement location. The mobile sensor system requires time to travel from one measurement location to another. Therefore, it might be better to choose a close measurement location with worse sensor-to-target geometry than one that is farther away but leads to more informative measurements. Such questions are discussed in the field of *sensor management*, which studies the optimization of reconfigurable sensor systems. Section 3.3 gives a brief overview over this field with a focus on sensor path planning.

## 3.1 Sensor data fusion

Even the measurements of modern, state-of-the-art sensors still have limitations:

- Measurements only provide incomplete information over the target state: for example, a radar does not measure the full velocity vector, and a DF sensor does not return the full position of a target.

- Measurements are noisy: the measurements are typically not exact, but influenced by electronic noise. For example, a DF sensor does not return the exact direction, but only an approximate measurement. An additional source of noise is the pose of the sensor platform. Especially in moving and flying platforms, the sensor position and orientation estimate by an *inertial navigation*

*system* (INS) might be noisy as well. This can happen due to accumulation errors when estimating the pose based on dead reckoning of acceleration sensors. Even GPS measurements are not perfectly accurate.

- Measurements might be false alarms: due to electrical noise in the receiver, a signal detection might be declared by pure chance. Active sensors, which emit their own signals, might wrongly declare unwanted reflections as targets. Such reflections could be the ground, buildings, chaff, or any other unwanted objects, summed up as *clutter*. Other sensors might be influenced by interfering signals in the environment, leading to false alarms.

- Measurements are not connected over time: if multiple targets are present in the environment, two measurements taken at different times do not necessarily indicate whether they originate from the same target. If measurements from multiple sensors are fused, this might even be unclear for measurements taken at the same time. This imposes the problem of *data association* [Koch, 2014, Ch. 3].

In the field of sensor data fusion, methods are studied to solve all of these limitations. In the context of this thesis, we are especially interested in the first two limitations: A sensor, in our case a direction finding sensor, only provides an incomplete measurement of the emitter position and this measurement is noisy.

Sensor data fusion addresses these problems by formulating the measurement process as a mathematical *measurement model*. Here, *measurement process* refers to the concrete physical operation and low level signal processing of the sensor. Based on the measurements and the measurement model, a belief about the current target state can be derived.

A series of beliefs about the state of the same target is known as *track* and the algorithm which produces the tracks is known as *tracker* or *localizer*. The term *tracker* emphasizes a change of the underlying ground truth over time, while the term *localizer* emphasizes a focus on the position part. In this thesis the term *localizer* is used because the target is stationary.

### 3.1.1 The Bayes filter

Most sensor data fusion algorithms use Bayesian statistics [Koch, 2014, Ch. 3; Bar-Shalom et al., 2001, Ch. 2] and model the information over the target state as a probability distribution over the target state space $\mathcal{X}^t$. This state space typically represents kinematic quantities of the target like its position and velocity, however can also contain properties like its radar cross section, emission frequency and power, or target type. Temporal progress is represented by predicting the probability distribution into the future, using a model of the target's behaviour. Following the temporal prediction, new measurements are integrated using Bayes' theorem and the measurement model.

A common assumption is that the Markov property holds for the targets, which means that the next target state only depends on the current target state. Then the state estimation can be modelled as a *hidden Markov model* (HMM), where the objective is to estimate the current target state based on the measurements. The measurement at *measurement step k*

$$\mathbf{z}_k = h_k(\mathbf{x}_k^t) + \mathbf{w}_k^m \tag{3.1}$$

is modelled by the *measurement function* $h_k$ of the true target state $\mathbf{x}_k$ with additive random noise $\mathbf{w}_k^m$. Equation (3.1) is called the *measurement model*. We commonly assume that the noise

$$\mathbf{w}_k^m \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k) \tag{3.2}$$

is distributed according to a zero-mean multivariate normal distribution with covariance $\mathbf{R}_k$, resulting in an *additive white Gaussian noise* model.

The subscript $k$ indicates that the measurement function and measurement covariance varies for different measurement steps. This is due to the potentially different sensor state $\mathbf{x}_k^s$ at different measurement steps. More explicit notations would be $h(\mathbf{x}_k^t, \mathbf{x}_k^s)$ or $h_{x_k^s}(\mathbf{x}_k^t)$, but these lead to cluttered formulas. As the sensor state is exactly known to the tracker, the measurement function $h_k$ at measurement step $k$ is known as well.

The tracker does not know the exact value of the target state $\mathbf{x}_k^t$. Instead, the belief about the target state is modelled as a random variable $\mathrm{X}^t$. The distribution of this random variable $p(\mathrm{X}^t \mid \mathbf{z}_{0:k})$ represents the knowledge about $\mathbf{x}_k^t$ after measurement step $k$, given the previously received measurements. Note that in addition to the measurements $\mathbf{z}_{0:k}$, the previous sensor states $\mathbf{x}_{0:k}^s$ leading to those measurements are known.

The behaviour of the target is modelled by a *process model*

$$\mathbf{x}_k^t = f^t(\mathbf{x}_{k-1}^t) + \mathbf{w}_{k-1}^p \tag{3.3}$$

with process noise $\mathbf{w}_k^p$. Similar to the transition function of an MDP, this function models a noisy state transition. However, the transition function of an MDP models the influence of an action on the state, while the actions of an observer in the typical sensor data fusion setting do not influence the behaviour of the target at all. In both cases, the process noise models uncontrollable changes in the state. In the process model of sensor data fusion, this also includes the unpredictable behaviour of the target and other mismatches between the simplified mathematical model and the true target dynamics.

The Bayes filter starts with the *initial belief* $b_{0|0}^t$, which is derived either from prior information or from a first measurement. When new measurements arrive, this belief is improved. The change of the belief due to new measurements is typically performed in two steps: *prediction* and *update*. The prediction step

$$p(\mathrm{X}_k^t \mid \mathbf{z}_{0:k-1}) = \int_{\mathcal{X}^t} p(\mathrm{X}_k^t \mid \mathrm{X}_{k-1}^t = \mathbf{x}) \cdot p(\mathrm{X}_{k-1}^t = \mathbf{x} \mid \mathbf{z}_{0:k-1})\, d\mathbf{x} \tag{3.4}$$

uses the process model to compute a probability distribution at the next measurement step, using only the previous belief $p(\mathrm{X}_{k-1}^t \mid \mathbf{z}_{0:k-1})$. The density $p(\mathrm{X}_k^t \mid \mathrm{X}_{k-1}^t)$ depends on the process model. The probability density (3.4) is also referred to as the *predicted belief* $b_{k|k-1}^t$. The subscript $k|k-1$ is common in the sensor data fusion literature and means that the belief is about the target state at measurement step $k$, but uses only measurements up to measurement step $k-1$. Equation (3.4) is also

known as the *Chapman-Kolmogorov equation* [Bar-Shalom et al., 2001, Ch. 10]. Given the predicted belief and a new measurement $\mathbf{z}_k$, Bayes' theorem can be used to compute the probability density

$$p(\mathrm{X}_k^t \mid \mathbf{z}_{0:k}) = \frac{p(\mathbf{z}_k \mid \mathrm{X}_k^t) \cdot p(\mathrm{X}_k^t \mid \mathbf{z}_{0:k-1})}{p(\mathbf{z}_k \mid \mathbf{z}_{0:k-1})} \,. \tag{3.5}$$

This probability density is referred to as *updated belief* $b_{k|k}^t$, where the subscript $k|k$ indicates that all measurements up to measurement step $k$ are used. As a short hand notation we also write

$$b_{k|k} = \mathrm{update}(b_{k|k-1}, \mathbf{z}_k, \mathbf{x}_k^s) \tag{3.6}$$

to represent the update step.

The probability density $p(\mathbf{z}_k \mid \mathrm{X}_k^t)$ is called the *measurement likelihood* and is also written as

$$\mathcal{L}_k(\mathbf{z}, \mathbf{x}) = p(\mathbf{z} \mid \mathrm{X}_k^t = \mathbf{x}) \,. \tag{3.7}$$

Note, that the measurement likelihood $\mathcal{L}_k$ of a measurement depends on the sensor state $\mathbf{x}_k^s$ at measurement step $k$.

As an example we combine two noisy bearing measurements $z_1$ and $z_2$, taken from different positions to localize a target. Both measurements indicate a measured angle from a known measurement location and are shown in Figure 3.2a. As they are a single angle, they are written as scalars $z_1, z_2$ and not as vectors $\mathbf{z}_1, \mathbf{z}_2$. Chapters 4, 5, and 6 also write bearing measurements as scalars. Instead of simply intersecting the measurements, the Bayes filter produces a probability density about possible target positions. We assume that the standard deviation of the measurement noise is known, for example due to a prior calibration of the sensor.

As in this example the target state $\mathbf{x}^t = (x^t, y^t)^\mathrm{T}$ is assumed to be stationary, a prediction step is not required. We assume that prior to any received measurements the target could be anywhere in the bounded target state space $\mathcal{X}^t = [-100\,\mathrm{m}, 100\,\mathrm{m}] \times [-100\,\mathrm{m}, 100\,\mathrm{m}]$ with uniform probability.

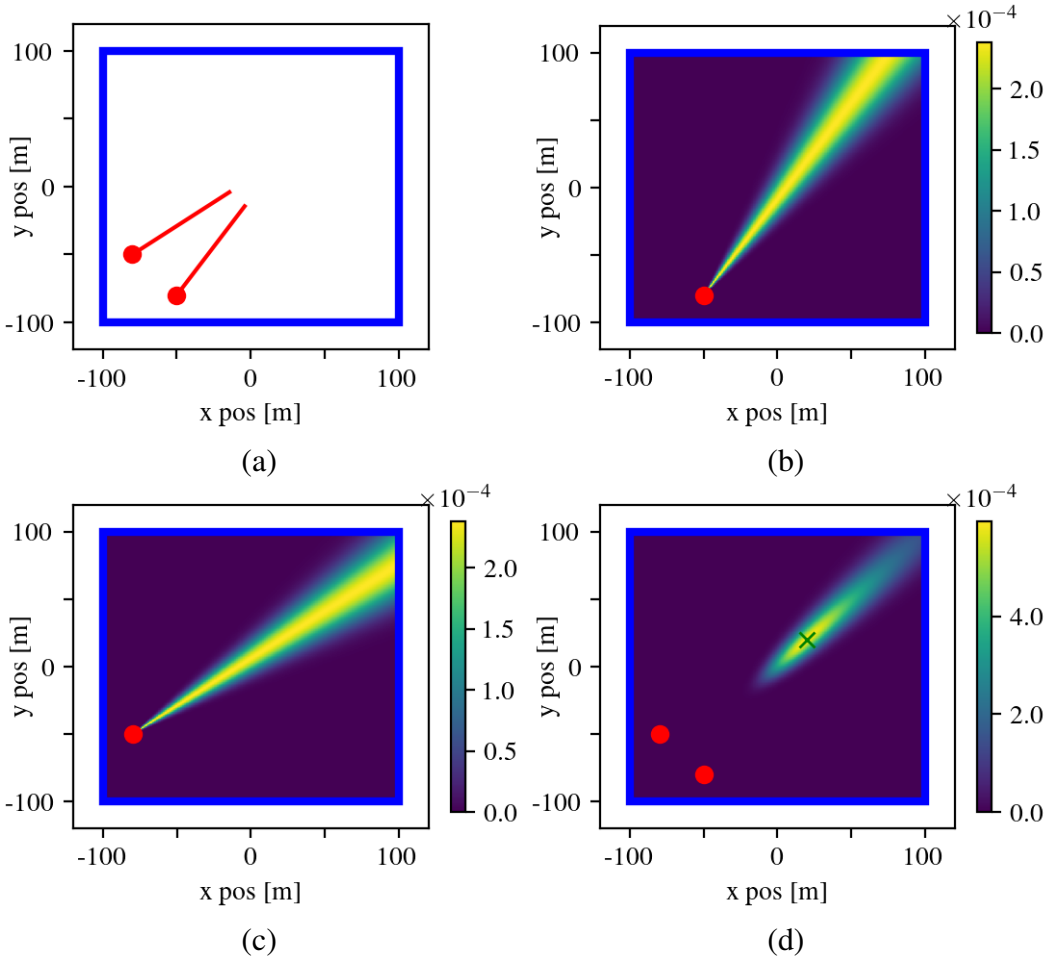



**Figure 3.2:** Combination of two bearing measurements $z_1$ and $z_2$. (a) The two measurements, (b) the updated belief $p(\mathrm{X}^t \mid z_1)$ using only the first measurement, (c) The updated belief $p(\mathrm{X}^t \mid z_2)$ using only the second measurement, (d) the fusion of both measurements $p(\mathrm{X}^t \mid z_1, z_2)$. The red circles denote the measurement locations, which are equal to the sensor states $\mathbf{x}_1^s$ and $\mathbf{x}_2^s$. The green cross in (c) shows the maximum a posteriori estimate of the target position. The blue rectangle indicates the target state space $\mathcal{X}^t$.

The belief is updated with the first measurement using Bayes' theorem, which results in

$$p(\mathrm{X}^t \mid z_1) = \frac{p(z_1 \mid \mathrm{X}^t) \cdot p(\mathrm{X}^t)}{p(z_1)} \,. \tag{3.8}$$

Figures 3.2b and 3.2c show the probability densities $p(\mathrm{X}^t \mid z_1)$ and $p(\mathrm{X}^t \mid z_2)$, evaluated on a grid. In Section 3.1.2 this grid will be explained in more detail. We note that because of the uniform prior, both densities are proportional to the measurement likelihood.

The integration of $z_1$ and $z_2$ is performed by making a Bayesian update of the belief after $z_1$ is known, with the new information $z_2$. The resulting distribution

$$p(\mathrm{X}^t \mid z_{1:2}) = \frac{p(z_2 \mid \mathrm{X}^t) \cdot p(\mathrm{X}^t \mid z_1)}{p(z_2 \mid z_1)} \tag{3.9}$$

is shown in Figure 3.2d. Note that in this figure the range of the colourbar is different than in (b) and (c), due to a higher concentration of this probability density. Note also that the computation of the density $p(\mathrm{X}^t \mid z_2)$ is not required and only shown for illustrative purposes in Figure 3.2b.

The described algorithm is known as the *Bayes filter* [Thrun et al., 2005, Ch. 2] and describes a generic algorithm for recursive state estimation. By specifying the way the probability densities are modelled, concrete algorithms can be derived. A well-known state estimation algorithm is the Kalman filter, which models all probability densities as multivariate normal and the measurement and process model as linear. Based on these assumptions the prediction and update step can be derived analytically.

### 3.1.2 Probability distributions

In this section, we discuss several choices for modelling the probability density $p(\mathrm{X}_k^t)$. We focus on two options, a multivariate Gaussian, and a discretization of the probability density on a grid. These are chosen because multivariate Gaussians are the most common solution and a discretization on a grid is especially viable for a stationary target. This is not an exhaustive list of options. Exemplary alternatives are modelling the probability density as a set of target state samples as done by

the particle filter [Gordon et al., 1993], or as Gaussian mixtures [Mušicki, 2009; Kronhamn, 1998]. As this thesis considers stationary targets for which no prediction step is required, this section focuses on the update step.

## The multivariate Gaussian distribution

With this distribution, we assume that the probability of the target state

$$\mathrm{X}_k^t \sim \mathcal{N}(\tilde{\mathbf{x}}_k^t, \tilde{\mathbf{P}}_k^t) \tag{3.10}$$

is distributed according to a multivariate normal distribution, centred on the *point estimate* $\tilde{\mathbf{x}}_k^t$ of the target state. A major advantage of this distribution is that it allows often for analytical solutions. For example, when the measurement model (3.1) and the process model (3.3) are both linear functions with additive Gaussian noise, prediction and update can be described analytically and result in the predicted and the updated belief being Gaussian as well. The resulting state estimation algorithm is known as the *Kalman filter* [Kalman, 1960; Koch, 2014, Ch. 3; Bar-Shalom et al., 2001, Ch. 5].

The prediction step of the Kalman filter

$$\tilde{\mathbf{x}}_{k|k-1}^t = \mathbf{F}\,\tilde{\mathbf{x}}_{k-1|k-1}^t \tag{3.11}$$

$$\tilde{\mathbf{P}}_{k|k-1}^t = \mathbf{F}\,\tilde{\mathbf{P}}_{k-1|k-1}^t\,\mathbf{F}^{\mathrm{T}} + \mathbf{Q} \tag{3.12}$$

uses a linear process model for the target, with $f^t(\mathbf{x}) = \mathbf{F}\mathbf{x}$ and Gaussian process noise $\mathbf{w}_k^p \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$. The predicted belief $b_{k|k-1}^t$ is the normal distribution $\mathcal{N}(\tilde{\mathbf{x}}_{k|k-1}^t, \tilde{\mathbf{P}}_{k|k-1}^t)$.

We now give an intuitive explanation of the update in the Kalman filter. The value $h_k(\tilde{\mathbf{x}}_{k|k-1})$ is called the *expected measurement*. The Kalman filter assumes that the measurement function $h_k$ is linear with $h_k(\mathbf{x}) = \mathbf{H}_k\mathbf{x}$. Similar as $h_k$, also $\mathbf{H}_k$ depends on the measurement step $k$ because the sensor state varies for different

$k$. Using $\mathbf{H}_k$, the filter computes the difference between the expected measurement $\mathbf{H}_k\tilde{\mathbf{x}}^t$ and the received measurement $\mathbf{z}_k$. This difference

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{H}_k\tilde{\mathbf{x}}^t_{k|k-1} \tag{3.13}$$

is called the *innovation*. A nonzero innovation means that the measurement is not the expected measurement, which might be due to two reasons. The point estimate could be wrong or the difference results from the measurement noise. To decide on how much the point estimate should be updated, we need to quantify how much the innovation is expected to vary. We expect a variance in the innovation due to the covariance $\mathbf{P}_{k|k-1}$ of the predicted belief, projected onto the measurement space, and the noise $\mathbf{R}_k$ of the measurement itself. From this we can compute the *innovation covariance*

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k \,. \tag{3.14}$$

Intuitively, the point estimate should change more if $\mathbf{P}_{k|k-1}$ is large, which indicates a high uncertainty in the predicted belief. It should change less, if we expect the innovation to vary a lot, which means $\mathbf{S}_{k+1}$ is large. This trade-off is captured by the *Kalman gain*

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}}\mathbf{S}_k^{-1} \tag{3.15}$$

which is a linear function that specifies how much the innovation in each measurement dimension should affect the point estimate. The Kalman gain is now used to update the point estimate of the target state and its covariance:

$$\tilde{\mathbf{x}}^t_{k|k} = \tilde{\mathbf{x}}^t_{k|k-1} + \mathbf{K}_k\mathbf{y}_k \tag{3.16}$$

$$\tilde{\mathbf{P}}^t_{k|k} = \tilde{\mathbf{P}}^t_{k|k-1} - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^{\mathrm{T}}. \tag{3.17}$$

Above, the intuition of the Kalman filter equations were introduced. These equations can be formally derived from the Bayes filter equations [Koch, 2014, Ch. 3]. Therefore, the Kalman filter corresponds exactly to the Bayes filter if the assumptions of normal distributed probability densities and linearity hold.

In many applications, for example with bearing measurements, range measurements, or combined bearing and range measurements, the measurement function $h_k$ of a Cartesian target state is not linear. In this case, we can compute the innovation (3.13) with the nonlinear function $h_k$

$$\mathbf{y}_k = \mathbf{z}_k - h_k(\tilde{\mathbf{x}}^t_{k|k-1}) \tag{3.18}$$

instead of the linear equation (3.13), and continue to use the remaining formulas with $\mathbf{H}_k = \frac{\partial h_k}{\partial \mathbf{x}}(\tilde{\mathbf{x}}^t_{k|k-1})$ being the Jacobian of $h_k$ evaluated at $\tilde{\mathbf{x}}^t_{k|k-1}$. In problems that require a prediction step, a similar linearization can be done if the process model is nonlinear. This algorithm is known as *extended Kalman filter* (EKF) [Bar-Shalom et al., 2001, Ch. 10]. Ideally, the measurement and process model should be linearized at the true target state $\mathbf{x}^t_k$. The EKF uses the point estimate $\tilde{\mathbf{x}}^t_{k|k-1}$ of the predicted belief for linearization, as this is the most likely target state before the measurement $\mathbf{z}_k$ is received. After the update, the point estimate $\tilde{\mathbf{x}}_{k|k}$ of the updated belief would be a better estimate of the true target state. The idea of the *iterated extended Kalman filter* (IEKF) is to iterate the update step multiple times to approximate the maximum a posteriori estimate numerically [Bell and Cathey, 1993; Bar-Shalom et al., 2001, Ch. 10].

For a given multivariate Gaussian distribution $\mathrm{X}^t \sim \mathcal{N}(\tilde{\mathbf{x}}^t, \tilde{\mathbf{P}}^t)$, the *Mahalanobis distance*

$$d^M(\mathbf{x}, \tilde{\mathbf{x}}^t, \tilde{\mathbf{P}}^t) = \sqrt{(\mathbf{x} - \tilde{\mathbf{x}}^t)^{\mathrm{T}} \left(\tilde{\mathbf{P}}^t\right)^{-1} (\mathbf{x} - \tilde{\mathbf{x}}^t)} \tag{3.19}$$

gives a normalized distance from the point $\mathbf{x}$ to the point estimate $\tilde{\mathbf{x}}^t$. The equation

$$d^M(\mathbf{x}, \tilde{\mathbf{x}}^t, \tilde{\mathbf{P}}^t) = \tau \tag{3.20}$$

specifies a hyperellipsoid, also called *confidence ellipsoid*, centred on $\tilde{\mathbf{x}}^t$ [Bar-Shalom et al., 2001, Ch. 3]. For the scalar case with $T = 1$, $\mathbf{P} = \sigma^2 \in \mathbb{R}^{1\times 1}$, $\mathbf{x} = x \in \mathbb{R}$ and $\tilde{\mathbf{x}}^t = \tilde{x}^t \in \mathbb{R}$, the confidence ellipsoid

$$\begin{aligned} \tau &= d^M(x, \tilde{x}^t, \sigma^2) && (3.21)\\ &= \sqrt{(x-\tilde{x}^t)^{\mathrm{T}} \frac{1}{\sigma^2}(x-\tilde{x}^t)} && (3.22)\\ &= \frac{|x-\tilde{x}^t|}{\sigma} && (3.23)\\ &\Leftrightarrow \\ x &= \tilde{x}^t \pm \tau\sigma && (3.24) \end{aligned}$$

corresponds to the $\tau\sigma$ confidence interval. A confidence ellipsoid according to (3.20) will therefore be called a $\tau\sigma$ confidence ellipsoid.

For the random variable $\mathrm{X}^t$, the distribution of $d^M(\mathrm{X}^t, \tilde{\mathbf{x}}^t)^2$ is chi-square distributed with $T$ degrees of freedom [Bar-Shalom et al., 2001, Ch. 1]. Therefore, the probability that $\mathrm{X}^t$ is in the volume

$$\mathcal{E}_\tau(\tilde{\mathbf{x}}^t, \tilde{\mathbf{P}}^t) = \{\mathbf{x} \in \mathbb{R}^T : d^M(\mathbf{x}, \tilde{\mathbf{x}}^t, \tilde{\mathbf{P}}^t)^2 \leq \tau^2\} \tag{3.25}$$

can be computed by the cumulative distribution of the chi-square distribution. The probabilities for $T = 1$ and $T = 2$ are given in Table 3.1.

**Table 3.1:** Probability that the target is in the confidence ellipsoid

| | $\tau = 1$ | $\tau = 2$ | $\tau = 3$ |
|---|---|---|---|
| $T = 1$ | 0.682 | 0.954 | 0.997 |
| $T = 2$ | 0.393 | 0.865 | 0.989 |

## Discrete distribution on a grid

Another option would be to discretize the target state space and represent the probability density as a discrete probability distribution over this discretized state space. A common discretization is in form of a grid, which discretizes each dimension of

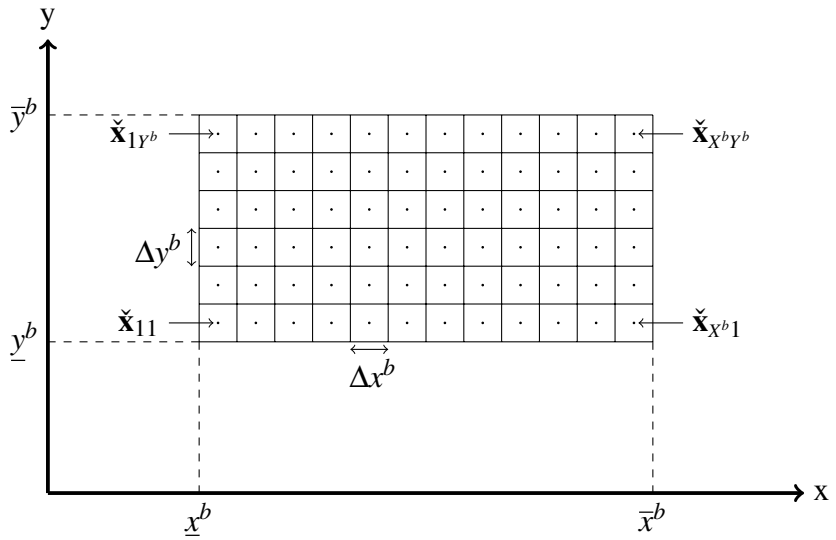



**Figure 3.3:** Definitions for a 2D grid.

the target state space in regular intervals. If the number of intervals per dimension is constant, the number of grid cells increases exponentially with the dimension of the target state space. This makes such a *grid-based Bayes filter* mostly useful if the number of dimensions is small, for example if only the two-dimensional position of a stationary and earth-bound target is estimated. The literature contains several uses of a grid-based Bayes filter, where either the position is assumed to be stationary [Cliff et al., 2015; Dressel and Kochenderfer, 2018] or the movement model does not require the estimation of velocity [Engin and Isler, 2020]. This algorithm is also called *histogram filter* [Thrun et al., 2005, Ch. 4].

Figure 3.3 shows a grid for a two-dimensional target state. We refer to a grid with $X^b$ cells in the $x$-direction and $Y^b$ in the $y$-direction as $X^b \times Y^b$ grid. The superscript $b$ indicates that the grid is used to represent a belief. This is to distinguish it from a grid representing an action space, later used in Chapter 4. To discretize the state, we need to restrict the target state space to certain limits within the target might possibly be located. These limits of the belief are denoted as $\underline{x}^b, \overline{x}^b, \underline{y}^b, \overline{y}^b$. We assume that the probability density inside each cell is constant and denote this value with $p_{kij}$, where $k$ stands for the measurement step and $ij$ for the cell index. The probability that the target is inside this cell is then $\Delta x_k^b \cdot \Delta y_k^b \cdot p_{kij}$, where $\Delta x^b, \Delta y^b$ are the width and height of the cell.

The update of the probability density with a measurement $\mathbf{z}_k$ is based on Bayes' theorem, where the measurement likelihood is evaluated at the cell centres $\check{\mathbf{x}}_{ij}$. The new probability density is given by

$$p_{kij} = \frac{1}{\xi} \cdot p_{[k-1]ij} \cdot \mathcal{L}_k(\mathbf{z}_k, \check{\mathbf{x}}_{ij}) \tag{3.26}$$

where

$$\xi = \Delta x_k^b \cdot \Delta y_k^b \cdot \sum_{i=1}^{X^b} \sum_{j=1}^{Y^b} p_{[k-1]ij} \cdot \mathcal{L}_k(\mathbf{z}_k, \check{\mathbf{x}}_{ij}) \tag{3.27}$$

is a normalizing factor, ensuring that

$$\sum_{i=1}^{X^b} \sum_{j=1}^{Y^b} \Delta x^b \cdot \Delta y^b \cdot p_{kij} = 1 \ . \tag{3.28}$$

Without $\Delta x_k^b \cdot \Delta y_k^b$ in (3.27) the normalization would lead to $p_{kij}$ being the probability of the target being in cell $ij$. By dividing also by $\Delta x_k^b \cdot \Delta y_k^b$, $p_{kij}$ represents the probability density in a cell.

The grid-based Bayes filter can also compute a point estimate $\tilde{\mathbf{x}}_k^t$, either by the expected value or the maximum a posteriori estimate. The expected value

$$\tilde{\mathbf{x}}_k^{t,ev} = \mathbb{E}\left[\mathrm{X}_k^t\right] \tag{3.29}$$

$$= \sum_{i=1}^{X^b} \sum_{j=1}^{Y^b} p_{kij} \cdot \Delta x^b \cdot \Delta y^b \cdot \check{\mathbf{x}}_{ij} \tag{3.30}$$

is the mean of all cell centres $\check{\mathbf{x}}_{ij}$, weighted by the probability mass in each cell. The *maximum a posteriori estimate*

$$\tilde{\mathbf{x}}_k^{t,map} = \check{\mathbf{x}}_{i^* j^*} \tag{3.31}$$

with

$$(i^*, j^*) = \underset{(i,j) \in \mathbb{I}}{\operatorname{argmax}} \ p_{kij} \tag{3.32}$$

is the centre of the cell with the highest probability density. Here $\mathbb{I} = \{1,...,X^b\} \times \{1,...,Y^b\}$ is the set of all cell indices. Similar to the Kalman filter notation, we call $h(\tilde{\mathbf{x}}_k^t)$ the expected measurement.

The probability density of the grid-based Bayes filter can be approximated with a Gaussian. For this, we use a point estimate $\tilde{\mathbf{x}}_k^t$ of the density, and approximate the covariance matrix of the Gaussian by

$$\tilde{\mathbf{P}}_k^t = \mathbb{E}\left[(\mathrm{X}_k^t - \tilde{\mathbf{x}}_k^t)(\mathrm{X}_k^t - \tilde{\mathbf{x}}_k^t)^{\mathrm{T}}\right] \tag{3.33}$$

$$= \sum_{i=1}^{X^b} \sum_{j=1}^{Y^b} p_{kij} \cdot \Delta x^b \cdot \Delta y^b \left(\check{\mathbf{x}}_{kij} - \tilde{\mathbf{x}}^t\right)\left(\check{\mathbf{x}}_{kij} - \tilde{\mathbf{x}}_k^t\right)^{\mathrm{T}} . \tag{3.34}$$

In the case that $\tilde{\mathbf{x}}_k^t$ is based on $\tilde{\mathbf{x}}_k^{t,ev}$, $\tilde{\mathbf{P}}_k^t$ corresponds to the covariance of $\mathrm{X}_k^t$, otherwise it is an approximation, centred on $\tilde{\mathbf{x}}_k$.

## Comparison

Figure 3.4 shows a comparison between the IEKF using a Gaussian distribution and the grid-based Bayes filter. The localizer is given a target state space of equally possible target positions and a single bearing, as shown in Figure 3.4a.

To create an initial Gaussian belief, the point estimate is set to the center of the scenario area and the covariance is chosen such that the $3\sigma$ confidence ellipsoid covers almost the whole area (see Figure 3.4b). An IEKF with 10 iterations is used to compute the updated belief, shown in Figure 3.4c. In this figure we can see some of the limitations of the Gaussian approach. Primarily, the form of the confidence ellipsoid does not correspond to what we would expect from a bearing measurement: a cone, which becomes wider with increasing distance due to the angular measurement noise. The ellipsoid form also leads to further inconsistencies. According to the probability density there is a nonzero probability that the target is outside of the scenario area. There is also a nonzero probability that the target is in the opposite direction of the bearing measurement, in other words behind the sensor, which would be very unlikely. Other positions, for example the upper left corner, are considered unlikely even though they are in line of the bearing.

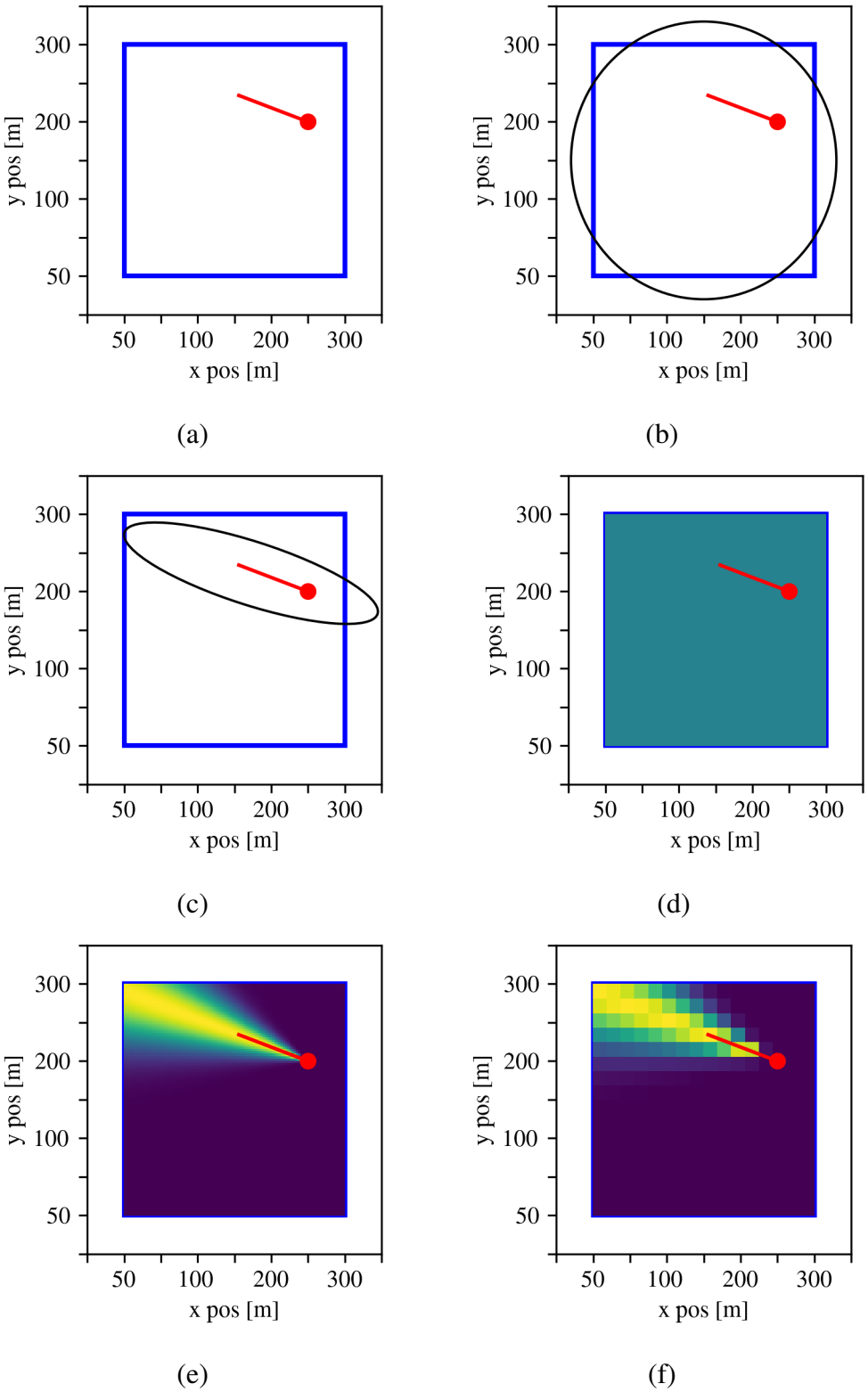



**Figure 3.4:** Comparison of the IEKF and the grid-based Bayes filter. The initial belief is a uniform distribution inside the target state space (blue). The setup and bearing measurement (red) is shown in (a). The Gaussian initial and updated belief ($3\sigma$ confidence ellipsoid) are shown in (b) and (c), respectively. The initial belief of the grid-based Bayes filter is shown in (d), the updated belief on a $100 \times 100$ grid in (e) and on a $16 \times 16$ grid in (f).

The grid-based Bayes filter minimizes these issues. While a Gaussian distribution cannot properly represent a uniform distribution, this is possible for a discrete distribution as shown in Figure 3.4d. Figure 3.4e shows the updated belief, which does not have the issues previously identified for the Gaussian. Naturally, the result of the grid-based Bayes filter becomes more inaccurate if the resolution of the grid is reduced (Figure 3.4f).

The advantage of the grid-based Bayes filter is that it is better suited to model the probability density resulting from a nonlinear measurement function and a constrained target state space. A disadvantage is the increased computation time. In this thesis we want to optimize the next sensing action on the basis of the current belief. Therefore, it is of importance that the updated belief corresponds as accurately as possible to the exact Bayesian update. We will therefore represent the belief in form of a grid-based Bayes filter. On the other hand, the later described path planning algorithm will compute many belief updates by sampled future measurements. This requires a computationally efficient implementation. Therefore, in Chapter 4 an adaptive grid-based Bayes filter is introduced, which successively increases its resolution when the target becomes better localized.

### 3.1.3 Fisher information and Cramér-Rao lower bound

Due to the measurement noise, a measurement does not provide perfect information about the target state. It is possible to quantify the available information contained in a measurement using the Fisher information. The information is based on the *score*

$$\mathrm{S}_k(\mathbf{x}_k^t) = \frac{\partial \log \mathcal{L}_k}{\partial \mathbf{x}}(\mathrm{Z}, \mathbf{x}^t) \tag{3.35}$$

which is defined as the gradient of the measurement log-likelihood that is evaluated at the true target state $\mathbf{x}^t$ [Cover and Thomas, 2006, Ch. 11]. The score is dependent on the random measurement Z, which makes it a random variable itself. It can be

shown that the expected value of the score is zero, therefore the covariance matrix of the score is given as

$$\mathbf{J}_k(\mathbf{x}^t) = \mathbb{E}\left[\left(\frac{\partial \log \mathcal{L}_k}{\partial \mathbf{x}}(\mathrm{Z}, \mathbf{x}^t)\right)\left(\frac{\partial \log \mathcal{L}_k}{\partial \mathbf{x}}(\mathrm{Z}, \mathbf{x}^t)\right)^{\mathrm{T}}\right] . \tag{3.36}$$

The covariance matrix of the score is called *Fisher information*. To stress the matrix form, in the case of $T > 1$, it is also called *Fisher information matrix* (FIM). Like every covariance matrix, the FIM is positive semi-definite.

An important application of the Fisher information is the *Cramér-Rao lower bound* (CRLB). It can be shown that the covariance of an unbiased point estimator E of $\mathbf{x}^t$, given the measurement Z is limited by

$$\mathrm{Cov}(\mathrm{E}) \geq \mathbf{J}(\mathbf{x}^t)^{-1} \tag{3.37}$$

which means that $\mathrm{Cov}(\mathrm{E}) - \mathbf{J}(\mathbf{x}^t)^{-1}$ is a positive semidefinite matrix [Cover and Thomas, 2006, Ch. 11]. Similar to the confidence ellipsoid (3.25), we can define a CRLB ellipsoid $\mathcal{E}_\tau(\mathbf{x}^t, \mathbf{J}^{-1})$, which allows to visualize the lower bound.

Intuitively, the CRLB means that with a high amount of Fisher information we can estimate $\mathbf{x}^t$ more accurately than with a low amount of Fisher information. When the true target state is constant, the Fisher information of multiple independent measurements can be added to compute the Cramér-Rao lower bound of an estimator using all those measurements.

Like for any square matrix, the inverse of $\mathbf{J}$ only exists if its determinant $|\mathbf{J}|$ is nonzero [Lang, 2004, Ch. 6]. In the case of $|\mathbf{J}| = 0$, the available information is not sufficient to perform a point estimate of the target state. We then call the target state *unobservable*.

We now discuss a special case which often appears in sensor data fusion. Suppose that the measurement is distributed according to the standard measurement model (3.1) with measurement function

$$h_k : \mathbb{R}^T \to \mathbb{R}^Z \tag{3.38}$$

and additive Gaussian measurement noise (3.2). This is equivalent to a multivariate Gaussian distribution

$$\mathrm{Z} \sim \mathcal{N}(h_k(\mathbf{x}^t), \mathbf{R}_k) \tag{3.39}$$

centred on $h_k(\mathbf{x}^t)$ with known covariance matrix $\mathbf{R}_k$ and leads to the measurement log-likelihood

$$\log \mathcal{L}_k(\mathbf{z}, \mathbf{x}) = \log \frac{1}{\sqrt{2\pi|\mathbf{R}_k|}} - \frac{1}{2}(\mathbf{z} - h_k(\mathbf{x}))^{\mathrm{T}} \mathbf{R}_k^{-1} (\mathbf{z} - h_k(\mathbf{x})) \,. \tag{3.40}$$

This leads to the score $\mathrm{S}_k(\mathbf{x}^t) \in \mathbb{R}^{1\times T}$ of

$$\begin{aligned}
\mathrm{S}_k(\mathbf{x}^t) &= \frac{\partial \log \mathcal{L}_k}{\partial \mathbf{x}}(\mathrm{Z}, \mathbf{x}^t) && (3.41)\\
&= -\frac{1}{2}\frac{\partial}{\partial \mathbf{x}}(\mathrm{Z} - h_k(\mathbf{x}^t))^{\mathrm{T}} \mathbf{R}_k^{-1} (\mathrm{Z} - h_k(\mathbf{x}^t)) && (3.42)\\
&= -\frac{1}{2} \cdot 2 \cdot (-1) \cdot \left(\mathrm{Z} - h_k(\mathbf{x}^t)\right)^{\mathrm{T}} \mathbf{R}_k^{-1} \frac{\partial h}{\partial \mathbf{x}}(\mathbf{x}^t) && (3.43)\\
&= (\mathrm{Z} - h_k(\mathbf{x}^t))^{\mathrm{T}} \mathbf{R}_k^{-1} \frac{\partial h}{\partial \mathbf{x}}(\mathbf{x}^t) && (3.44)\\
&= (\mathrm{Z} - h_k(\mathbf{x}^t))^{\mathrm{T}} \mathbf{R}_k^{-1} \mathbf{H}_k \,. && (3.45)
\end{aligned}$$

Here $\frac{\partial h_k}{\partial \mathbf{x}}(\mathbf{x}^t) = \mathbf{H}_k \in \mathbb{R}^{Z\times T}$ is the Jacobian of $h_k$, evaluated at the true target state $\mathbf{x}^t$. Equation (3.43) follows from the multivariate chain rule, using that $\frac{\partial}{\partial \mathbf{x}} \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x} = 2\mathbf{x}^{\mathrm{T}} \mathbf{A}$ for any symmetric matrix $\mathbf{A}$. Then the Fisher information is

$$\begin{aligned}
\mathbf{J}_k(\mathbf{x}^t) &= \mathbb{E}\left[\mathrm{S}_k(\mathbf{x}^t)^{\mathrm{T}} \mathrm{S}_k(\mathbf{x}^t)\right] && (3.46)\\
&= \mathbb{E}\left[\left((\mathrm{Z} - h_k(\mathbf{x}^t))^{\mathrm{T}} \mathbf{R}_k^{-1} \mathbf{H}_k\right)^{\mathrm{T}} (\mathrm{Z} - h_k(\mathbf{x}^t))^{\mathrm{T}} \mathbf{R}_k^{-1} \mathbf{H}_k\right] && (3.47)\\
&= \mathbb{E}\left[\mathbf{H}_k^{\mathrm{T}} \left(\mathbf{R}_k^{-1}\right)^{\mathrm{T}} (\mathrm{Z} - h_k(\mathbf{x}^t))(\mathrm{Z} - h_k(\mathbf{x}^t))^{\mathrm{T}} \mathbf{R}_k^{-1} \mathbf{H}_k)\right] && (3.48)\\
&= \mathbf{H}_k^{\mathrm{T}} \left(\mathbf{R}_k^{-1}\right)^{\mathrm{T}} \mathbb{E}\left[(\mathrm{Z} - h_k(\mathbf{x}^t))(\mathrm{Z} - h_k(\mathbf{x}^t))^{\mathrm{T}}\right] \mathbf{R}_k^{-1} \mathbf{H}_k && (3.49)\\
&= \mathbf{H}_k^{\mathrm{T}} (\mathbf{R}_k^{-1})^{\mathrm{T}} \mathbf{R}_k \mathbf{R}_k^{-1} \mathbf{H}_k && (3.50)\\
&= \mathbf{H}_k^{\mathrm{T}} \mathbf{R}_k^{-1} \mathbf{R}_k \mathbf{R}_k^{-1} \mathbf{H}_k && (3.51)\\
&= \mathbf{H}_k^{\mathrm{T}} \mathbf{R}_k^{-1} \mathbf{H}_k \,. && (3.52)
\end{aligned}$$

Equation (3.49) follows from the linearity of the expected value and (3.50) from the fact that this expected value is the covariance of the measurement, which due to the measurement model (3.39) is $\mathbf{R}_k$. Step (3.51) follows because $\mathbf{R}_k$ is symmetric and invertible, and therefore $\mathbf{R}_k^{-1}$ is also symmetric.

We could interpret a Gaussian belief $\mathcal{N}(\tilde{\mathbf{x}}, \tilde{\mathbf{P}})$ also as a single measurement of the true state with $h_k$ being the identity function and $\mathbf{H}_k$ the identity matrix. The Fisher information of such a measurement would be $\tilde{\mathbf{P}}^{-1}$. Therefore, the inverse $\tilde{\mathbf{P}}^{-1}$ of a covariance matrix $\tilde{\mathbf{P}}$ is also called the *information matrix*.

### 3.1.4 Uncertainty metrics

Intuitively, it is better if the belief distribution is more concentrated and we therefore have a clearer idea on what are the more likely values of the state. In this section, we define metrics that quantify whether one belief is more or less accurate than another belief. Such a metric is a function

$$\mathcal{B} \to \mathbb{R} \tag{3.53}$$

that maps a belief to a real number.

We begin by the *expected root mean squared error* (expected RMSE). For this we first define the *mean squared error* (MSE). Given a point estimate $\tilde{\mathbf{x}}^t = (\tilde{x}_1, \ldots, \tilde{x}_T)$ of the true target state $\mathbf{x}^t = (x_1, \ldots, x_T)$, the MSE is given by

$$\mathrm{MSE}(\tilde{\mathbf{x}}^t, \mathbf{x}^t) = \sum_{i=1}^{T} (\tilde{x}_i - x_i)^2 \,. \tag{3.54}$$

The root of the MSE

$$\sqrt{\mathrm{MSE}(\tilde{\mathbf{x}}^t, \mathbf{x}^t)} = \sqrt{\sum_{i=1}^{T} (\tilde{x}_i - x_i)^2} \tag{3.55}$$

$$= \|\tilde{\mathbf{x}}^t - \mathbf{x}^t\|_2 \tag{3.56}$$

is called root mean squared error (RMSE). This value corresponds to the Euclidean distance between the point estimate and the true target state. If the target state

consists of different parts with different dimensions, it is common to state the RMSE of those parts separately. For example, if the target state $\mathbf{x}^t = (x, y, \dot{x}, \dot{y})$ consists of a position and velocity part, one could compute a separate RMSE for each part. The expected RMSE

$$\mu(\tilde{\mathbf{x}}^t, b^t) = \mathbb{E}\left[\|\tilde{\mathbf{x}}^t - \mathrm{X}^t\|_2\right] \tag{3.57}$$

is defined for a belief as the expected value of the RMSE, where $p(\mathrm{X}^t)$ is given by the belief $b^t$. Commonly, $\tilde{\mathbf{x}}^t$ is chosen based on the belief as well, for example the maximum a posteriori estimate or the expected value. For the grid-based Bayes filter, the expected RMSE is computed by

$$\mu(\tilde{\mathbf{x}}^t, b) = \sum_{i=1}^{X^b} \sum_{j=1}^{Y^b} p_{ij} \cdot \Delta x^b \cdot \Delta y^b \, \|\tilde{\mathbf{x}}^t - \check{\mathbf{x}}_{ij}\|_2 \, . \tag{3.58}$$

Similarly, an expected MSE $\mathbb{E}\left[\|\tilde{\mathbf{x}}^t - \mathrm{X}^t\|_2^2\right]$ can be defined.

The *Shannon entropy* is a metric that gives the information content of a probability distribution. For a discrete random variable $\mathrm{X}^t$, it is defined as

$$\mathcal{H}^S(b) = -\sum_{\mathbf{x} \in \mathbb{X}} \mathcal{P}(\mathrm{X}^t = \mathbf{x}) \cdot \log \mathcal{P}(\mathrm{X}^t = \mathbf{x}) \tag{3.59}$$

where $\mathbb{X}$ would be the set of all possible outcomes with positive probability [Cover and Thomas, 2006, Ch. 2] and $\mathcal{P}(\mathrm{X}^t = \mathbf{x})$ denotes the probability that $\mathrm{X}^t$ takes the value $\mathbf{x}$. In our case, $\mathbb{X}$ would correspond to a discrete target state space. We can use the Shannon entropy with the discrete distribution of the grid-based Bayes filter, identifying each cell with its centre, $\mathbb{X} = \{\check{\mathbf{x}}_{ij} : p_{ij} > 0\}$, and $\mathcal{P}(\mathrm{X}^t = \check{\mathbf{x}}_{ij}) = \Delta x^b \cdot \Delta x^b \cdot p_{ij}$. The Shannon entropy is always positive.

For continuous probability density functions with support set $\mathcal{S} = \{\mathbf{x} \in \mathcal{X}^t \mid p(\mathrm{X}^t = \mathbf{x}) > 0\}$, the *differential entropy* [Cover and Thomas, 2006, Ch. 8] is defined as

$$\mathcal{H}^D(b) = -\int_{\mathcal{S}} p(\mathrm{X}^t = \mathbf{x}) \log p(\mathrm{X}^t = \mathbf{x}) \mathrm{d}\mathbf{x} \, . \tag{3.60}$$

Contrary to the Shannon entropy, the differential entropy can also be negative. This can be seen for example with the differential entropy of a $T$-dimensional multivariate Gaussian [Cover and Thomas, 2006, Ch. 8]

$$\mathcal{H}^D(\mathcal{N}(\tilde{\mathbf{x}},\tilde{\mathbf{P}})) = \frac{1}{2}\log\left((2\pi e)^T|\tilde{\mathbf{P}}|\right) \tag{3.61}$$

which is negative if the determinant of the covariance $\tilde{\mathbf{P}}$ becomes sufficiently small. Contrary to the other equations in this thesis, in (3.61) $\pi \approx 3.14159$ refers to the mathematical constant instead of a policy and $e \approx 2.71828$ is Euler's number.

The following metrics are specific for beliefs consisting of multivariate Gaussians. They are typically a function of the covariance matrix $\tilde{\mathbf{P}}^t$. As the covariance matrix is bounded by the inverse Fisher information matrix $\mathbf{J}^{-1}$, they are also commonly used to quantify the size of the FIM.

A common uncertainty metric for Gaussians is the determinant, with

$$\left(\tilde{\mathbf{x}},\tilde{\mathbf{P}}^t\right) \mapsto \left|\tilde{\mathbf{P}}^t\right| \ . \tag{3.62}$$

This metric is based on the fact that the volume of the $1\sigma$ confidence ellipsoid is given by

$$\mathrm{vol}(\mathcal{E}_1(\tilde{\mathbf{x}}^t,\tilde{\mathbf{P}}^t)) = \sqrt{|\tilde{\mathbf{P}}^t|}\cdot\mathrm{vol}(\mathcal{B}_T) \ , \tag{3.63}$$

and therefore proportional to the square root of the determinant [Pronzato and Pázman, 2013, Ch. 5]. Here

$$\mathrm{vol}(\mathcal{B}_T) = \frac{\pi^{T/2}}{\Gamma(T/2+1)} \tag{3.64}$$

is the volume of the $T$-dimensional unit ball. In (3.64) $\Gamma$ refers to the gamma function and again $\pi$ is the mathematical constant. When applying this metric to the Fisher information, the volume of the $1\sigma$ CRLB ellipsoid is given by

$$\mathrm{vol}(\mathcal{E}_1(\mathbf{x}^t,\mathbf{J})) = \frac{1}{\sqrt{|\mathbf{J}|}}\cdot\mathrm{vol}(\mathcal{B}_T) \ . \tag{3.65}$$

A higher value of the determinant of $\mathbf{J}$, and a lower value of the determinant of $\tilde{\mathbf{P}}^{\mathbf{t}}$ lead to a smaller volume of the confidence ellipsoid. Another common uncertainty metric for Gaussians is the trace, with

$$\left(\tilde{\mathbf{x}}, \tilde{\mathbf{P}}^t\right) \mapsto \mathrm{tr}\left(\tilde{\mathbf{P}}^t\right) \ . \tag{3.66}$$

This metric is an analytic form of the expected MSE

$$\mathrm{tr}(\tilde{\mathbf{P}}) = \mathbb{E}\left[\|\tilde{\mathbf{x}}^t - \mathrm{X}^t\|_2^2\right] \tag{3.67}$$

for multivariate Gaussians [Bar-Shalom et al., 2001, Ch. 1].

Finally, the largest eigenvalue $\lambda_1^e$ of $\tilde{\mathbf{P}}^{\mathbf{t}}$ is proportional to the major semiaxis of the uncertainty ellipsoid squared. Therefore, the metric

$$\left(\tilde{\mathbf{x}}, \tilde{\mathbf{P}}^t\right) \mapsto \lambda_1 \tag{3.68}$$

considers a belief more accurate than another if its major axis is smaller. The analogue for the Fisher information matrix is to take the smallest eigenvalue, with a longer one being better.

## 3.2 Direction finding sensors

Direction finding (DF) sensors measure the direction of arrival (DOA) of a signal. There exist direction finding sensors for almost all types of signals, for example acoustic direction finding, underwater acoustic or passive sonar, or optical sensors. In this thesis we focus on radio signals.

Several technologies exist to implement DF sensors. An example would be the use of array antennas [Oispuu, 2013]. An array antenna consists of a spatial arrangement of several individual antennas. The DOA of the incoming signal leads to phase differences at the individual antennas, which can be measured. The advantage of an array antenna consists in an almost instantaneous measurement of the

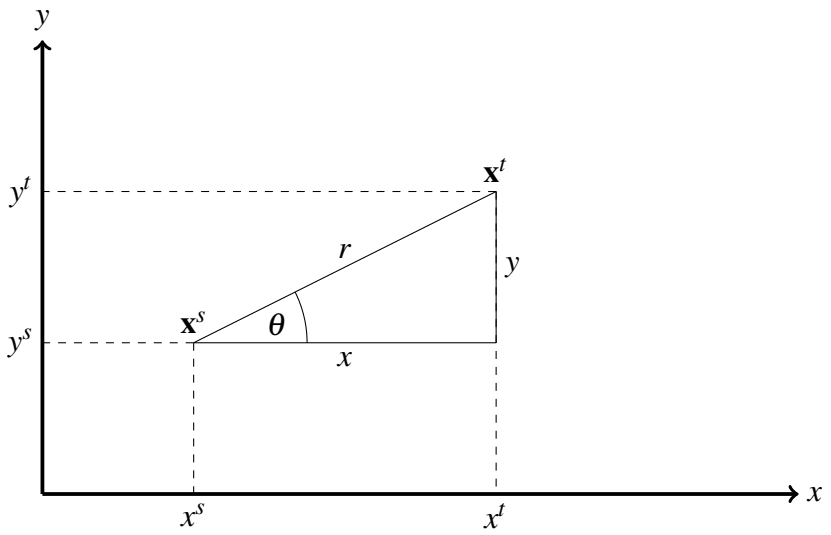



**Figure 3.5:** Definitions for a bearing measurement. The target bearing $\theta$ of a target with state $\mathbf{x}^t = (x^t, y^t)^{\mathrm{T}}$ is measured by a DF sensor at position $\mathbf{x}^s = (x^s, y^s)^{\mathrm{T}}$. The relative position of the target with respect to the sensor is $(x, y)^{\mathrm{T}}$.

signal direction. However, array antennas also come with disadvantages like high cost, complex signal processing, and calibration needs.

Another solution is the use of a directional antenna. A directional antenna has a direction dependent antenna gain, often with a clearly distinguished main lobe. Therefore, by rotating the antenna, one can determine the origin of the signal. The disadvantage of this approach is that due to the required rotation of the antenna, a single measurement takes more time. It is also not feasible to find the DOA of a signal that is significantly shorter than the time required to rotate the antenna. Advantages of this approach are that it can be implemented cheaper and requires less complex signal processing and calibration. Due to the prospect of having a cheap and less complex localization technique, in this thesis we will implement a sensor system based on this second approach.

In this thesis we assume that the elevation component of the direction is not available and we can only measure the horizontal angle towards the target. The *target bearing* $\theta$ is defined as the counter-clockwise angle between the positive $x$-axis and the line between sensor and target. This angle can be seen in Figure 3.5.

In the experiments, the $x$-axis will point towards east and the $y$-axis towards north. The measurement function is given by

$$h_k^{\mathrm{DF}}(\hat{\mathbf{x}}) = \operatorname{atan2}(\hat{y} - y_k^s, \hat{x} - x_k^s) \tag{3.69}$$

for the target position $\hat{\mathbf{x}} = (\hat{x}, \hat{y})^{\mathrm{T}}$ and the sensor position $\mathbf{x}_k^s = (x_k^s, y_k^s)^{\mathrm{T}}$. The function atan2 is the four-quadrant arctangent function. This geometry is visualized in Figure 3.5. The measurement

$$z_k = h_k^{\mathrm{DF}}(\mathbf{x}^t) + w_k^m \tag{3.70}$$

consists of the true bearing with additive Gaussian noise $w_k^m \sim \mathcal{N}(0, \sigma^2)$ with standard deviation $\sigma$.

### 3.2.1 Fisher information

We now want to quantify the amount of information available in a single bearing measurement. For this, we take an axis-aligned coordinate system centred on the sensor position $\mathbf{x}^s$. As the position of the sensor $\mathbf{x}^s$ is known exactly, the information gained over the relative target position $(x, y)^{\mathrm{T}} = \mathbf{x}^t - \mathbf{x}^s$ corresponds to the information gained about its absolute position $(x^t, y^t)^{\mathrm{T}}$. This gives us the measurement equation

$$h(x, y) = \operatorname{atan2}(y, x) \tag{3.71}$$

whose Jacobian is

$$\mathbf{H} = \left( \frac{\partial h}{\partial x}, \frac{\partial h}{\partial y} \right) \tag{3.72}$$

$$= \left( \frac{y}{x^2 + y^2}, \frac{-x}{x^2 + y^2} \right) \tag{3.73}$$

$$= \frac{1}{r^2} \cdot (y, -x) \ . \tag{3.74}$$

The last equation uses the distance $r = \sqrt{x^2 + y^2}$ between target and sensor. The covariance of the measurement noise $\mathbf{R} = \sigma^2 \in \mathbb{R}^{1 \times 1}$ is a scalar.

Using equation (3.52), the Fisher information of a bearing measurement is given by

$$\mathbf{J}(x,y) = \mathbf{H}^{\mathrm{T}}\mathbf{R}^{-1}\mathbf{H} \tag{3.75}$$

$$= \frac{1}{\sigma^2} \cdot \mathbf{H}^{\mathrm{T}}\mathbf{H} \tag{3.76}$$

$$= \frac{1}{r^4\sigma^2} \begin{pmatrix} y^2 & -yx \\ -xy & x^2 \end{pmatrix} . \tag{3.77}$$

In absolute coordinates $\mathbf{x}^t = (x^t, y^t)$ for the target and $\mathbf{x}^s = (x^s, y^s)$ for the sensor, the Fisher information is given by

$$\mathbf{J}(\mathbf{x}^t, \mathbf{x}^s) = \mathbf{J}(x^t - x^s, y^t - y^s) . \tag{3.78}$$

Using the substitution $x = r\cos\theta$ and $y = r\sin\theta$, the Fisher information is given by

$$\mathbf{J}(\theta, r) = \frac{1}{r^2\sigma^2} \begin{pmatrix} \sin^2\theta & -\sin\theta\cos\theta \\ -\sin\theta\cos\theta & \cos^2\theta \end{pmatrix} \tag{3.79}$$

$$= \frac{1}{r^2\sigma^2} \begin{pmatrix} \sin^2\theta & -\frac{1}{2}\sin 2\theta \\ -\frac{1}{2}\sin 2\theta & \cos^2\theta \end{pmatrix} . \tag{3.80}$$

We can draw several conclusions from the Fisher information. The determinant of the Fisher information

$$|\mathbf{J}(x,y)| = \frac{1}{r^4\sigma^2} \cdot (y^2x^2 - (-xy)^2) = 0 \tag{3.81}$$

is zero, which shows that the matrix is not invertible and therefore, the target state is unobservable by a single measurement. Equation (3.80) shows that the information content of a measurement decreases quadratically with the distance $r$ between target and sensor. We can use some edge cases to further our understanding of the Fisher information matrix: If $\theta = 0^\circ$ or alternatively $y = 0$, zero information is available about the x-coordinate of the target, as the x-axis is parallel to the target bearing. The same is true if $\theta = 90^\circ$ or $x = 0$, where no information about the y-coordinate is gathered.

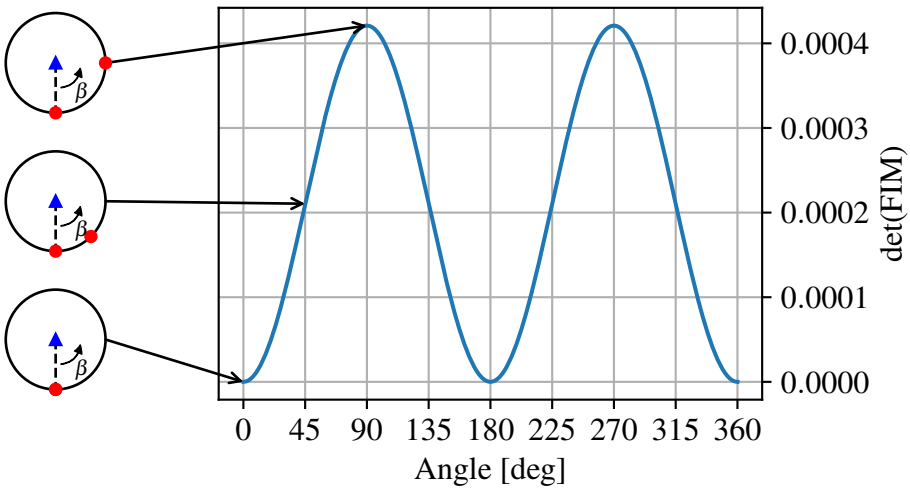



**Figure 3.6:** Information for different geometries of two measurement locations (red) with the same distance to the target (blue triangle).

### 3.2.2 Sensor-to-target geometry

We call the position of the sensor when taking the measurement $\mathbf{z}$ the *measurement location* of $\mathbf{z}$ and the relative positioning of sensor and target when taking the measurement the *sensor-to-target geometry*. The previous section already showed that the position of the target is unobservable from a single measurement. This section discusses how the sensor-to-target geometry for multiple measurements influences the received information about the target position. This gives an intuition on how measurement locations should be selected to achieve a good localization. Similar analyses have been presented in the literature [Hoffmann and Tomlin, 2010; Bishop et al., 2007, 2010].

We begin by characterizing the information content of two measurements with the same standard deviation $\sigma$, taken at equal distance $r$ to the target. Using the determinant metric, the total information content of both measurements is given by

$$\left|\mathbf{J}(\theta_0, r) + \mathbf{J}(\theta_1, r)\right| . \tag{3.82}$$

Figure 3.6 shows a numerical evaluation of (3.82) for different measurement locations, parameterized by an angle $\beta$. These measurement locations lead to the target

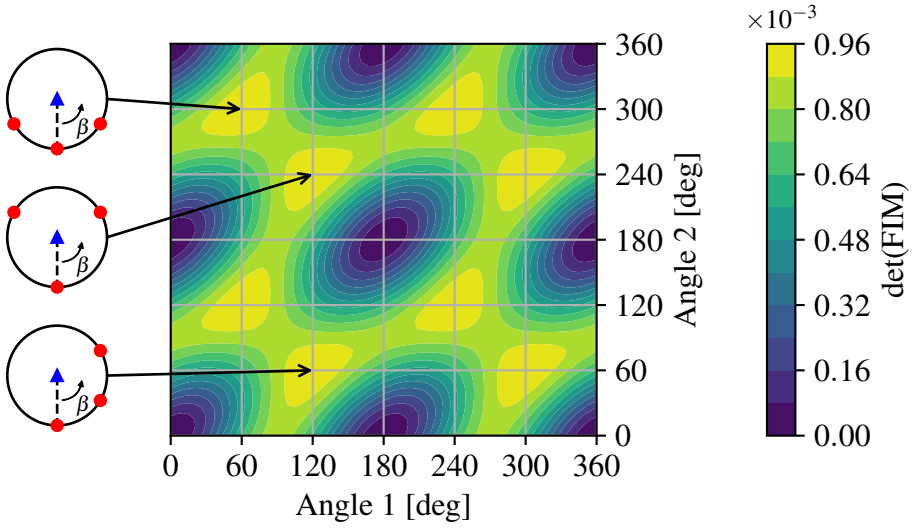



**Figure 3.7:** Information for different geometries of three measurement positions (red) with the same range to the target (blue).

bearings $\theta_0 = -90°$, $\theta_1 = \theta_0 + \beta$. Maximal information is achieved at $\beta = 90°$ and $\beta = 270°$, which corresponds to the intuition that the optimal sensor-to-target geometry for two measurements consists of a perpendicular cross-bearing. Analytically, we can find based on (3.79) that

$$|\mathbf{J}(0,r) + \mathbf{J}(\beta,r)| = \frac{1}{r^2\sigma^2}\sin^2\beta \tag{3.83}$$

which corresponds to the graph in Figure 3.6. Figure 3.7 shows a similar graph for the case of three measurements, with target bearings $\theta_0 = -90°$, $\theta_1 = \theta_0 + \beta_1$ and $\theta_2 = \theta_0 + \beta_2$. Here the optimal separation would be either 60° or 120° between the measurement locations.

Equation (3.80) shows that the Fisher information matrix is proportional to the scalar $1/\sigma^2$ and therefore, this factor can be moved out of the determinant in (3.82). For this reason the maxima of (3.82) stay the same with different $\sigma$ and optimal geometries are not dependent on the standard deviation of the measurement noise.

This section gave an intuition about how optimal sensor-to-target geometries look like. For this, we assumed an ideal situation where the ground truth position of the target is available, measurements can be taken at arbitrary positions, and no fur-

ther constraints limit the problem. The next section discusses sensor management, which is the problem of how to use a sensor optimally. This includes not only the ideal sensor-to-target geometries as described above but also the problem of how to achieve them in a realistic setting, without these idealized assumptions.

## 3.3 Sensor management

In the measurement function $h_k$, the index $k$ indicates that the function might be different in different measurement steps due to a change in the sensor state $\mathbf{x}_k^s$. A different sensor state leads to a different measurement process or a different sensor-to-target geometry, and therefore different measurements.
Examples for a sensor state are:

- The waveform that is used by an active sensor. For example, an active electronically steered array radar can change many parameters of its waveform: Its pulse repetition frequency, carrier frequency, number of pulses, beam direction, and more. The parameters of the currently used waveform would be part of the sensor state $\mathbf{x}_k^s$. The corresponding sensor management problem is discussed in the context of *cognitive radar* in the literature [Haykin, 2006; Greco et al., 2018; Gurbuz et al., 2019].

- A passive RF receiver might listen for signals on multiple frequency bands [Clarkson, 2011; Apfeld et al., 2016]. If it can only listen at a single frequency band for a given time, the sensor state $\mathbf{x}_k^s$ would correspond to the frequency to which the receiver is tuned.

- The measurement could be created by multiple sensors, where at each time only a single one or a subset can be used [Li et al., 2009]. The state $\mathbf{x}_k^s$ would specify which sensors are used in measurement step $k$. In this case $\mathbf{x}_k^s$ would more properly be referred to as the state of the sensor system, but would otherwise have the same semantics.

- The position and orientation of the sensor [Ragi and Chong, 2013; Hammel et al., 1989]. In many situations, measurements are relative to the sensor

position, as for example bearing or range measurements, or if the sensor has a restricted field of view. In this case $\mathbf{x}_k^s$ would contain the position of the sensor.

Also combinations of these examples are possible, for example in a radar the waveform and its position are both important. The sensor state influences the measurement process or sensor-to-target geometry and therefore leads to a better or worse performance of the whole sensor system.

A sensor is called *reconfigurable* if we can change its state before taking measurements. In the field of *sensor management*, methods are researched to do this in an automatic way. Based on the previous measurements and the resulting belief (see Section 3.1), a *sensor controller* changes the sensor state to improve the performance of the whole sensor system. We call the output of the sensor controller a *sensing action*. On the one hand, this is done to draw a connection to sequential decision processes, presented in Chapter 2. On the other hand, this stresses that the sensor controller cannot directly modify the sensor state $\mathbf{x}_{k+1}^s$ arbitrarily for the next measurement, but only indirectly by a sensing action $\mathbf{a}_k$. For example, the sensor controller might change the sensor position only indirectly by changing the motion vector of the platform. This would restrict the sensor state $\mathbf{x}_{k+1}^s$ to those states reachable from $\mathbf{x}_k^s$. The sensor controller leads to a closed loop between sensor, tracker, and sensor controller. This loop is shown in Figure 3.8.

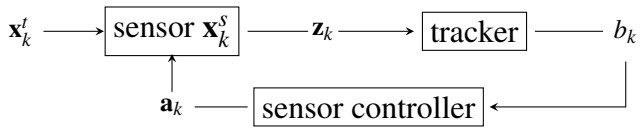



**Figure 3.8:** Sensor management loop.

### 3.3.1 Sensor management as a POMDP

In the language of approximate dynamic programming, we can identify the concatenation of target state and sensor configuration

$$\mathbf{x}_k = \begin{pmatrix} \mathbf{x}_k^t \\ \mathbf{x}_k^s \end{pmatrix} \tag{3.84}$$

with the state known from Markov decision processes in Section 2.2. We refer to it as the *joint state* to distinguish it from the target state and sensor state. The sensing action $\mathbf{a}_k \in \mathcal{A}$ changes only the sensor state and does not affect the target state. The transition function of the joint state

$$\mathrm{X}_{k+1} = f^a(\mathbf{x}_k, \mathbf{a}_k, \mathrm{W}_k^p) = \begin{pmatrix} f^t(\mathbf{x}_k^t) \\ f^s(\mathbf{x}_k^s, \mathbf{a}_k) \end{pmatrix} + \begin{pmatrix} \mathrm{W}_k^p \\ \mathbf{0} \end{pmatrix} \tag{3.85}$$

is given by the process model of the target state (3.3) and a suitable *sensor transition function* $f^s$.

We assume in (3.85) that the transition of the sensor state is deterministic. A nondeterministic transition could be added by adding noise to the sensor state. Noise could be used to model parts of the sensor state that are controlled by different processes. For example, in the control of an airborne radar, it could be that the radar is controllable, but the kinematic state of the aircraft is not. Also, purely random effects as for example wind might influence the sensor state.

The belief $b_k$ is a probability distribution over the joint state space $\mathcal{X} = \mathcal{X}^t \times \mathcal{X}^s$, with

$$b_k(\mathbf{x}) = p(\mathrm{X} = \mathbf{x}) \ . \tag{3.86}$$

Commonly, the sensor state is observable, which we can represent using the Dirac delta distribution. The belief is then given by

$$b_k(\hat{\mathbf{x}}) = p(\mathrm{X}^t = \hat{\mathbf{x}}^t) \cdot \delta(\mathbf{x}^s - \hat{\mathbf{x}}^s) \tag{3.87}$$

with $\hat{\mathbf{x}} = (\hat{\mathbf{x}}^t, \hat{\mathbf{x}}^s)$. As discussed in Section 2.2.4, this is merely a notational tool to stay consistent with the idea that the belief is a probability distribution over the state space. In a real implementation, the belief $b_k^t$ over the target state is typically represented as a probability distribution, and the sensor state $\mathbf{x}_k^s$ would be directly available.

The sensor controller is a policy

$$\pi : \mathcal{B} \longrightarrow \mathcal{A} \tag{3.88}$$

that maps the current belief $b_k \in \mathcal{B}$ to the sensing action $\mathbf{a}_k \in \mathcal{A}$. Due to the assumption of an observable state, the policy can also be given as

$$\pi : \mathcal{B}^t \times \mathcal{X}^s \longrightarrow \mathcal{A} \tag{3.89}$$

a mapping from the belief about the target state and the current sensor state to the next sensing action.

As the sensor controller creates a new sensing action after each received measurement, we can consider the decision step $k$ from Section 2.2 and the measurement step $k$ from Section 3.1 the same. In the remainder of this thesis, we will therefore consider those two as the same index. In particular, the measurement step $k$ refers to the execution of the measurement model, which creates the measurement $\mathbf{z}_k$ and an updated belief $b_k$. The decision step $k$ refers to the execution of the sensor controller, based on this updated belief $b_k$.

Having the same number of decision steps and measurement steps is valid if the frequency of new measurements is sufficiently slow or the sensor controller sufficiently fast. For high frequency sensors or slow sensor controllers, it would be required to execute the sensor controller at a slower rate than the measurement processing. In this case, $k$ would refer to the decision step of the sensor controller, $b_k$ would be the current belief at this decision step, and $\mathbf{a}_k$ would change the sensor state for multiple measurement steps. However, the closed loop structure would stay the same.

As in Section 2.2, a cost function

$$c_k : \mathcal{B} \times \mathcal{A} \longrightarrow \mathbb{R} \tag{3.90}$$

will be defined. This cost function is a mathematical formulation of the sensing task. There is significant research about cost functions in sensor management, which is briefly discussed in the next section.

### 3.3.2 Approaches to sensor management

Sensor management algorithms can be divided into two main classes: rule-based and optimization-based. Rule-based algorithms are fixed decision procedures. An example would be to schedule a radar dwell every four seconds or to move a platform with a DF sensor perpendicular to the last measured bearing. Optimization-based approaches consider the decision as an optimization problem and find the optimal next sensing action by solving this problem.

Most of the current research is focused on optimization-based approaches. It follows the general idea that a sensor system designer should not specify the sensing actions directly but instead the sensing task which the sensor system should perform. Typically, the sensing task is specified using the cost function (3.90). In this thesis, we classify optimization-based sensor management according to three different aspects: The cost function, the mathematical model of the problem, and the planning horizon.

#### Cost function

A common classification of cost functions is into *task-driven* and *information-driven* functions [Aoki et al., 2011a; Kreucher et al., 2005].

A task-driven cost function is a cost function that refers to a task-specific metric of the sensor system. Such a metric could for example be the RMSE between the point estimate of the target position and the ground truth [Aoki et al., 2011a], number of lost tracks, or target detection probability [Kreucher et al., 2005]. The cost function then consists of the expected value of this metric, given the current belief. An example would be the expected RMSE as discussed in Section 3.1.4.

In comparison, an information-driven cost function is based on a metric from information theory [Cover and Thomas, 2006]. Information theory formalizes the abstract idea that some distributions have a higher information content than others.

This is done by using concepts like the entropy, for example in [Ryan et al., 2007]. The entropy was previously discussed in Section 3.1.4 and leads to the cost function

$$c(b_k^t, \mathbf{x}_k^s, \mathbf{a}_k^s) = \mathcal{H}^D(b_k^t) \; . \tag{3.91}$$

One can see that the sensing action and sensor state have no direct influence on the cost function. Their influence on the cost is only indirect, as the chosen sensing action will change the sensor state in the next measurement step $k+1$ and therefore the measurement $\mathbf{z}_{k+1}$, belief $b_{k+1}$, and the resulting cost.

Similarly, the determinant, trace, or largest eigenvalue have been used in many sensor management applications, either applied to the covariance of the belief or to the Fisher information matrix [Leung et al., 2006; Oshman and Davidson, 1999; Doğançay, 2012; Vander Hook et al., 2015]. As the Fisher information requires the true target state, this is typically approximately computed for the point estimate of the target state. Other often-used information-driven cost functions are the Kullback Leibler divergence and the Renyi divergence [Cover and Thomas, 2006, Chapter 8 and 17], for example in [Katsilieris et al., 2012; Ristic et al., 2010]. Those divergences quantify the difference between probability distributions. In the context of sensor management, they are applied to the known predicted belief distribution $b_{k|k-1}$ and the new belief distribution after measurement step $b_{k|k}$. A larger difference between those two distributions means that the belief about the target state has improved.

The main difference between a task-driven and an information-driven cost function is the intention. A task-driven cost function quantifies a value that the user of a sensor system cares about. It is a task-specific criterion and refers directly to the sensing task. For example, a user of a sensor system might care about having a small error to the ground truth or a small number of false alarms. A major difficulty of task-driven approaches lies in the fact that often multiple, potentially conflicting, criteria are important for a sensing task, and either trade-offs or a surrogate cost function needs to be defined. Information-theoretic cost functions are intended to be a generic way to specify such surrogate cost functions. The idea is that by

minimizing for example the entropy, and therefore increase the general information content of the belief, the sensor controller will also optimize the task-specific metrics a user is concerned about. Whether information-theoretic cost functions can truly serve as such an *universal proxy* is a topic of ongoing research [Aoki et al., 2011b].

## Mathematical model of the problem

The mathematical model can be *deterministic* or *stochastic*. As discussed in Section 3.1, the measurement process itself is always a stochastic process due to measurement noise and partial observability. Similarly, the behaviour of the target is unknown and provides another source of uncertainty, modelled with a process model. However, this uncertainty is not always considered in the sensor management algorithm.

A deterministic model approximates this uncertainty by setting random variables to fixed values. For example, a common approximation is to replace noise terms by zero and assume the target to be at the centre of the estimated probability density. This is for example done in [Cliff et al., 2015]. Similarly, the *nominal belief-state optimization* (NBO) method [Miller et al., 2009; Ragi and Chong, 2013] creates a stochastic model of the problem, but performs the planning on a deterministic model. This model is created by replacing the future beliefs by nominal ones, which result from the process and measurement model under the assumption of zero process and measurement noise.

In comparison, a sensor controller that uses stochastic models explicitly considers the different outcomes of at least some of those random variables. For example, a stochastic model could provide different measurements based on multiple realizations of the measurement noise or different possible target states based on the current belief. If the model is correctly chosen, this can lead to a more accurate problem description than a deterministic approximation. In some cases, this requires a careful selection of the probability densities. While the measurement noise can often be derived accurately from calibration measurements of the sensor, the process noise in the process model (3.3) is often more a rough approximation of

the manoeuvrability of the target and the mismatch between model and reality, than a true description of the behaviour of the target.

An example for a stochastic sensor controller for the problem of sensor path planning is given in [Hernandez, 2004]. Here the sensor controller computes the value of the cost function in the next state based on Monte Carlo sampling. Then, an adaptive quadrant search is made to search for the action which minimizes the expected value. This algorithm can be generalized to a policy with an $H$-step planning horizon, with the cost of an exponential increase in computing time. In [Hernandez, 2004] results for a 2-step planning horizon were shown.

In [Trémois and Le Cadre, 1999], the Smallwood-Sondik algorithm [Smallwood and Sondik, 1973] for POMDPs was used to solve a discretized multi-step sensor path planning problem for a direction finding sensor. However, due to the computational constraints associated with the solution of the POMDP, it was required to restrict the state space to 25 positions and four possible observations. This is consistent with the general difficulty of solving POMDPs with offline algorithms.

The policy rollout method, described in the previous chapter, reduces the computational requirements by using a base policy to determine the future actions. It has been applied to sensor management applications [Chong et al., 2008], for example to the activation of sensors in a sensor network [He and Chong, 2004, 2006], sonar ping optimization [Saksena and Wang, 2008] and UAV control [Sarunic and Evans, 2014].

## Planning horizon

The *planning horizon* describes the number of decision steps considered after the current action. A major distinction is between *myopic* and *nonmyopic* approaches. A myopic approach considers only the effect of the next action and corresponds to a planning horizon of one. A nonmyopic approach also considers the effect of actions further in the future. Examples for myopic approaches are [Doğançay, 2012] and [Hoffmann and Tomlin, 2010] which both optimize the next movement step of multiple UAVs for sensor path planning and [Kershaw and Evans, 1994] which optimizes the next waveform used in a radar. Examples for nonmyopic approaches

are [Hammel et al., 1989] that provides sensor path planning for a single platform, [Leung et al., 2006] for multiple platforms, and [Li et al., 2009] which considers the activation of nodes in a sensor network.

Especially with a stochastic model of the problem, there is a qualitative difference between myopic and nonmyopic approaches. If the process model and measurement model is deterministic, a planning horizon greater than one corresponds to finding an optimal fixed sequence of sensing actions. However, if those models are nondeterministic, such an optimal fixed sequence might not necessarily exist. Instead, the optimal sequence of actions might be dependent on the future realizations of the measurement noise and process noise.

The size of a useful planning horizon is also dependent on the specifics of the sensor system and sensing task. An example would be a slow moving robot with a limited field of view that needs to detect stationary targets in an area. For such a sensor system, it makes sense to plan a complete trajectory ahead, as it is very costly to go back to a measurement location later. On the other hand, a radar is a very agile sensor, which often needs to track very agile targets. It is therefore difficult to consider each possible future target behaviour, making a long planning horizon less useful. But while it would be costly for a robot to move back to a previous measurement location, the nearly instantaneous steering of the radar dwell allows for a measurement without large additional cost, once this becomes necessary.

This observation can similarly be stated for the abstract POMDP mentioned above. The higher the process noise $\mathrm{W}^p$ of the process model for the target is, the less use has a long planning horizon. Similarly, a long planning horizon is of less use, the easier it is for the sensor manager to change the sensor state $\mathbf{x}^s$ to an arbitrary value. If multiple sensing actions are required to set the sensor state to a specific value, or a different order of sensing actions leads to different costs, a higher planning horizon is useful. This is often the case in the sensor path planning problem.

### 3.3.3 Sensor path planning

A subproblem of sensor management is path planning for mobile sensor systems. Previously, Section 3.2.2 described that the gathered information about the target state is dependent on the measurement locations. A mobile sensor system can adapt its position to optimize the sensor-to-target geometry. This is not only relevant for DF sensors, but for range and range-bearing sensors as well. The capacity of an RF sensor to detect a target also diminishes typically with range, as the energy of the signal is spread out on a sphere with a surface proportional to the distance squared. In addition, obstacles in the environment might block the signal or the sensor might only be able to receive a signal at certain angles. In this case, the field of view (FOV) of the sensor is limited. In all of these situations, an active control of the sensor position can help to achieve the sensing task.

We name the problem of optimizing the paths of a platform to improve a sensing task *sensor path planning*. The restriction of improving a sensing task means that this definition excludes problems where a sensor is used in a mobile sensor system but decisions about the path are not made to improve the sensing task. For example, navigation tasks where the sensor is used to avoid obstacles do not fall under this definition. We also distinguish the term from *shortest path search* and *motion planning*. In shortest path search the goal is to find the shortest path between two nodes in a graph. Such problems often appear in navigation tasks and can be solved using graph algorithms, as for example the Dijkstra algorithm [Cormen et al., 2001, Ch. 24]. Motion planning is the problem of finding the best path for a mobile platform in a continuous space to a goal state, where the platform is potentially limited by motion constraints [Karaman and Frazzoli, 2011; LaValle, 2006].

Problems which fall under the umbrella term of sensor path planning have been considered in the literature in different contexts. For example in the problem of autonomous map building [Kollar and Roy, 2008; Charrow et al., 2015], a robot with a laser range finder is given the task to optimize its path to improve the completeness and accuracy of the map. The works by [Cai and Ferrari, 2009; Zhang et al., 2009] consider a mine clearing application, where a mobile robot with a range lim-

ited sensor needs to classify already localized targets. Coverage path planning is the problem of quickly traversing an area such that every position is visited at least once. This problem appears if a sensor needs to search for targets in an area larger than its FOV. An application is for example underwater mine clearing [Paull et al., 2013].

In this thesis the focus is on sensor path planning for a DF sensor to localize an emitter. We will call a sensor controller for the sensor path planning problem a *path planner*, implying from the context that it is about sensors. As previously seen in Section 3.2.2, the localization accuracy of an emitter with multiple DF measurements is highly dependent on the sensor-to-target geometry. The localization accuracy is also dependent on the behaviour of the target. A trajectory that localizes a stationary target well, might have a poor performance when the target moves. The target state might even be unobservable in this case. Common classes of targets that are studied in the literature are stationary targets and linearly or piecewise-linearly moving targets.

A stationary target could for example be a ground-based radar which should be localized, a stationary interference source, or a signal beacon. A piecewise-linearly moving target could correspond to a radar equipped aircraft flying a straight course and is also common for localizing ships, for example by their radar or communication signals, or their acoustic emissions using a passive sonar. Based on these assumptions, sensor path planning is concerned with finding the optimal trajectory to localize the target.

### Typical sensor trajectories

The following paragraphs discuss typical trajectories appearing in the literature. In all of these scenarios the ground truth position of the target is assumed to be available and the Fisher information is used to optimize the trajectory, similar as in Section 3.2.2. To compute a sensor path online, the point estimate of the target state can be used. The cost function is the negative determinant of the Fisher information matrix, therefore a minimization of the cost corresponds to a maximization of the determinant. The optimization is made using the BFGS algorithm, and the gradi-

ents are computed using the *CasADi* library [Andersson et al., 2019]. The sensor state $\mathbf{x}_k^s = (x_k^s, y_k^s)^{\mathrm{T}}$ consists of the sensor position and the action $a \in \mathbb{R}$ is the next direction of travel. The sensor transition function is given by

$$x_{k+1}^s = x_k^s + \cos(a) \cdot v^s \cdot \Delta t \tag{3.92}$$

$$y_{k+1}^s = y_k^s + \sin(a) \cdot v^s \cdot \Delta t \ . \tag{3.93}$$

Note that the action space consists of $\mathbb{R}$, even though sin and cos are both periodic functions. This is because BFGS is an unconstrained optimization algorithm. Manoeuvring does not lead to a cost in the sensor transition function (3.93), the cost only depends on the information gained during the trajectory, as measured by the determinant of the Fisher information. The initial Fisher information is a zero matrix and the BFGS iteration is started by a zero-vector. The standard deviation of a measurement is $\sigma = 3^\circ$ for all trajectories.

We first consider the following problem: given a finite horizon scenario of $K = 10$ time steps and a stationary target, what is the trajectory such that the uncertainty about the target position after the last time step is minimized? Note, that the uncertainty in all steps $k < K$ is not important in this formulation. This problem can be solved by deterministic lookahead with a planning horizon of $H = K = 10$. The only cost is a final cost after the last time step. Figure 3.9a shows several trajectories for different velocities of the platform $v^s = 25\,\mathrm{m/s}, 50\,\mathrm{m/s}, ..., 200\,\mathrm{m/s}$. The trajectories are similar to the ones presented in [Hammel et al., 1989] and [Oshman and Davidson, 1999], and show the optimal trajectory for a given ratio between travel distance and target distance. The travel distance during the planning horizon $H$ is given by $v^s \cdot H \cdot \Delta t$, with platform speed $v^s$, planning horizon length $H$ and the *time step* $\Delta t = 1\,\mathrm{s}$ which is the interval between successive decision steps. This distance corresponds to the length of the trajectories in Figure 3.9. The numbers 0.125 to 1.0 at the end of trajectories indicate the ratio $\hat{r} = v^s \cdot H \cdot \Delta t / r^t$, where $r^t = 2000\,\mathrm{m}$ is the initial distance between platform and target. This ratio indicates how far the platform can travel relative to the distance to the target. If $\hat{r}$ is small, the optimal sensor path is to move almost perpendicular to the target such that the target can

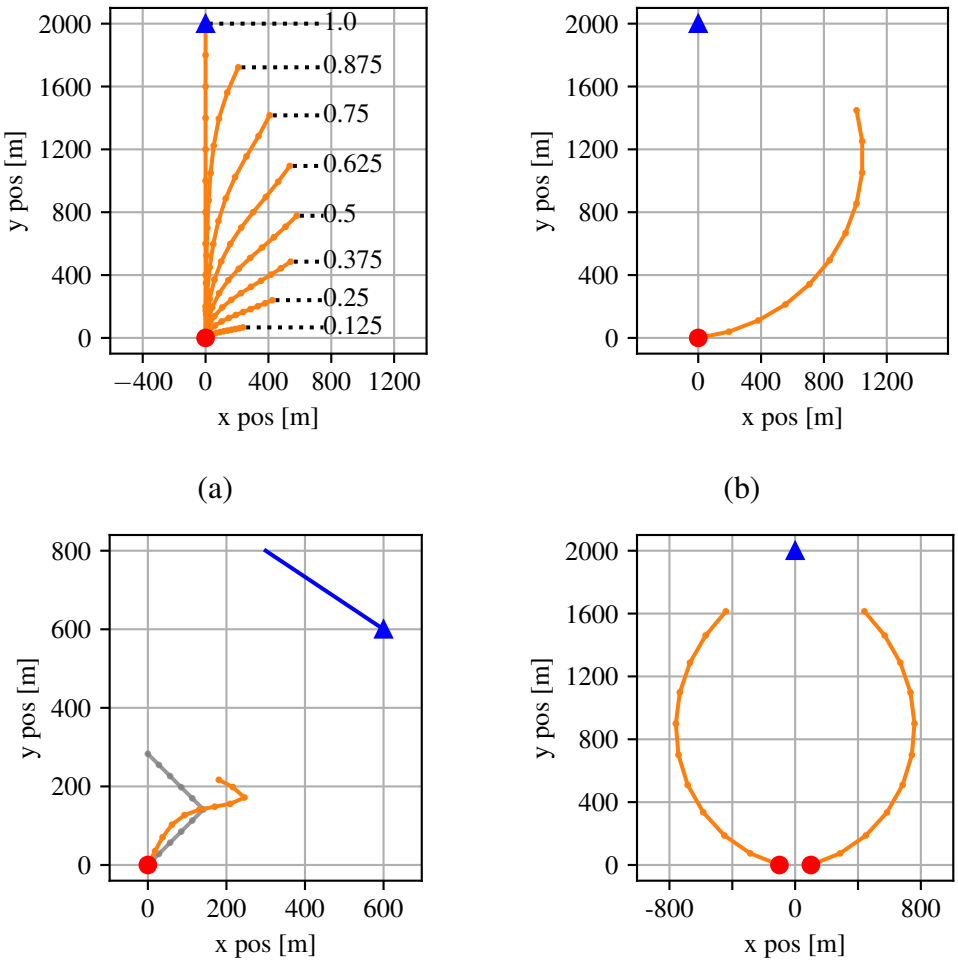



**Figure 3.9:** Typical paths for localization with bearing measurements. Red indicates the starting position of the sensor platform, the blue triangle the starting position of the target and orange the trajectory of the platform. (a) Different paths to localize a static target by a nonmyopic path planner. The numbers give the ratio between the distance the platform can travel during the planning horizon and the initial distance between platform and target. (b) Trajectory resulting from a myopic path planner to localize a static target. (c) Trajectory to localize a linearly moving target. The path planner is nonmyopic and the optimization algorithm was initialized with the grey trajectory. (d) Myopic path planner for two platforms and a stationary target.

be measured with different bearings. As $\hat{r}$ becomes larger, it becomes better to first reduce range and then move perpendicular. These two components, reducing the range and observing the target with different bearings, follow directly from the properties of the Fisher information shown in Section 3.2.2. If $\hat{r}$ approaches 1, the optimal decision becomes to travel directly to the target. As the Fisher information is proportional to $1/r^2$ with distance $r$, in the limit as $r \to 0$, it tends to infinity.

In comparison, Figure 3.9b shows the trajectory resulting from a myopic planning algorithm with a velocity of $v^s = 200\,\mathrm{m/s}$ and $K = 10$ time steps and a planning horizon of $H = 1$. It can be seen that it moves almost perpendicular towards the target, as it focuses on the immediate information gain. It also reduces the range, but slower than the nonmyopic path planner in Figure 3.9a.

Figure 3.9c shows a typical trajectory to localize a linearly moving target, for example a ship or a nonmaneuvering aircraft. When the target is nonstationary, observability becomes more complex [Nardone and Aidala, 1981; Fogel and Gavish, 1988]. For a linearly moving target, the sensor must move on a trajectory that performs manoeuvrers, which means that the platform must accelerate. The trajectory in Figure 3.9b consists of multiple segments, with the last one being almost perpendicular towards the target bearing. Such sensor paths are typical for localizing linearly moving targets [Passerieux and Van Cappel, 1998; Le Cadre and Laurent-Michel, 1999]. This trajectory was also computed using deterministic lookahead with $H = K = 10$ and only a final cost. Different to the other trajectories it required a nonzero initial Fisher information $\mathbf{J}_0 = \mathbf{I}_{4\times4} \cdot 10^{-10}$ and the BFGS algorithm was initialized with a trajectory guaranteeing observability. This initial trajectory is shown in grey in the Figure. The velocity of the platform was $v^s = 40\,\mathrm{m/s}$.

Sensor path planning has also been performed for multiple DF sensors, mounted on multiple platforms [Spletzer and Taylor, 2003; Hoffmann and Tomlin, 2010; Grocholsky et al., 2003]. In this case, the position of the target can be estimated directly by intersection, which makes observability a lesser concern. The resulting trajectories typically spread out such that the angle between the target and the platforms remains constant and they similarly reduce the range towards the tar-

get. As can be seen from the discussion in Section 3.2.2, maximal information is achieved by keeping a separation of 90° in the case of two platforms and either 60° or 120° in the case of three platforms. Figure 3.9d shows the trajectories resulting from a myopic path planner for two platforms with $v^s = 200\,\mathrm{m/s}$. Here, the action space is the joint steering $\mathbf{a} \in \mathbb{R}^2$ and the sensor state space $\mathcal{X}^s = \mathbb{R}^4$ the joint state of both platform positions. The trajectories spread out first, and afterwards converge on a 90° angle.

### Sensing actions with different durations

The trajectories discussed above make a standard assumption in sensor path planning, which is that the duration of an individual sensing action is constant and the platform moves for a limited distance between those sensing actions. When this is the case, the decision steps $k$ are based on a regular discretization of time with a time step $\Delta t$. Then, the distance the platform travels between the decision steps is limited by $v^s \cdot \Delta t$.

The fact that $\Delta t$ is constant and independent of the sensing action is an important idea because it allows myopic and receding horizon path planners to only consider the improvement of the posterior beliefs. For example, if an action $\mathbf{a}$ leads to a less costly posterior belief than another action $\mathbf{a}'$, it makes sense for a myopic path planner to chose this action. However, this is only true if the actions take the same time. If instead $\mathbf{a}'$ would have been much shorter in time, maybe an action sequence $\mathbf{a}', \mathbf{a}''$ would have been better than $\mathbf{a}$ alone. This assumption of constant time steps is implicitly made by almost all myopic and receding horizon path planners, but only rarely stated explicitly, for example in [Doğançay, 2012].

The assumption of constant time steps corresponds to a sensor system whose measurements occur independently from the movement of the sensor platform as they are not affected by it. This is a well justified assumption for many applications. For example, the measurements generated by an array-based DF sensor on an aircraft occur at regular intervals, independent of the movement.

However, with the progress of robotics and ever cheaper, smaller, and more available mobile platforms, several demonstrations of small mobile sensor sys-

tems with DF sensors have been performed [Dressel and Kochenderfer, 2018; Venkateswaran et al., 2013; Cliff et al., 2015; Vonehr et al., 2016; Vander Hook et al., 2015]. These systems do not use array antennas because of the higher cost, sophisticated processing requirements and harder calibration of this sensor type. Therefore, they often have different sensing characteristics as classically assumed. In particular it can be the case that the platform needs to stay stationary during a measurement to determine the direction of arrival with a directional antenna [Cliff et al., 2015; Vonehr et al., 2016; Vander Hook et al., 2015] or that it has different sensing characteristics during motion [Graefenstein et al., 2009]. Then a single sensing action consists of a movement phase of variable duration and a measurement phase. This leads to the duration of a sensing action not being constant any more.

Sensor path planning for such sensor systems has been demonstrated, however typically with a limited planning horizon and without taking into account the uncertainty of the measurement process. The work in [Cliff et al., 2015] describes a myopic path planner, which bases the next measurement decision on the results of the nominal measurement without taking travel costs into account. The path planner in [Vander Hook et al., 2015] selects as the next two measurement locations those which would be optimal if the emitter were exactly placed at the closest position on the $3\sigma$ confidence ellipsoid.

This thesis develops a path planner for a sensor system which needs to be stationary during the measurement process. The algorithm takes into account travel times as well as the measurement duration and minimizes the time until the target is localized. This task-driven cost function was chosen because it encodes a direct benefit for a user and is directly observable in an experiment. In general, nonmyopic planning is advantageous in the sensor path planning problem because physical movement of the sensor is not instantaneously revertible. This is especially true in the considered sensor system because measurements are not at constant time steps. Instead, the platform requires a significant amount of travel time before the next measurement is performed. Therefore, the path planner is nonmyopic and plans

until the end of the localization process. The path planner considers the uncertain current target state and outcome of future measurements. The next chapter introduces the path planner and compares it with path planners from the literature [Cliff et al., 2015; Vander Hook et al., 2015]. In Chapter 5, the path planner will be improved and made more efficient. Finally, in Chapter 6 an experimental evaluation is described.

# Chapter 4

# Policy Rollout for Sensor Path Planning

In this chapter, a path planner for localizing a stationary RF emitting target with a mobile UAV is described. The UAV is equipped with a directional antenna and can take a bearing measurement towards the target by rotating around its own vertical axis. The contents of this chapter have previously been published in [Hoffmann et al., 2019] and [Hoffmann et al., 2021]. Figure 4.1 visualizes the problem setting. The red lines are measurements of the target bearing, taken at different locations. The UAV moves on the straight path (blue) between two measurement locations.

As described in Section 3.2, when localizing a target with bearing measurements, the sensor-to-target geometry has a large influence on the accuracy of the result. A sensor system as described here has the additional challenge that the planner faces a trade-off: The measurement process and movement are both time intensive. Therefore, the path planner needs to decide whether the time is better spent by optimizing the sensor-to-target geometry or by taking an additional measurement.

Compared to more complex sensor systems, for example those with array-based sensors which can move and measure at the same time, such a sensor system has the advantage of being simpler and potentially lighter.

The first section of this chapter performs a comparison with existing work. The second section creates a mathematical formulation of the problem. The problem is described in form of decision steps, where each decision fixes the next sensing ac-

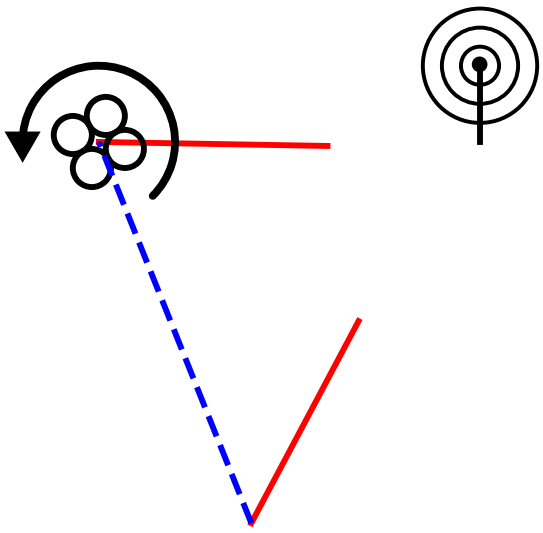


**Figure 4.1:** Visualization of the measurement process. By rotation around its own vertical axis, the UAV produces a bearing measurement (red). Afterwards, it moves to the next measurement location. The trajectory of this movement is shown in blue. At the next measurement location it then takes a new bearing measurement.

tion. The available sensing actions are the possible measurement locations, to which the UAV will fly to take a measurement. Therefore, different sensing actions lead to different travel times and have different durations. Section 4.3 describes a path planner, based on the policy rollout principle. This rollout path planner solves the problem by simulating the future development of the joint state and belief for multiple sensing actions, under the assumption that future actions are determined by a simple base policy. A simulative evaluation of the path planner is performed in Section 4.4, based on two scenarios. The first scenario evaluates the influence of a specific target position at different fixed positions. In the other scenario, the target position is sampled from a probability distribution to determine the average performance of the path planner. It can be seen that the rollout path planner performs

well, and on average leads to a faster localization than other path planners. The last section performs a conclusion of this chapter.

## 4.1 Comparison with existing work

Sensor systems that alternate between movement and taking a bearing measurement have been considered multiple times in the literature. Chapter 6 contains a literature survey about sensor systems of this type that have been demonstrated experimentally. This section describes path planning algorithms for such systems.

### 4.1.1 Mobile sensor systems with stationary measurements

In [Tokekar et al., 2011], three path planners are proposed. All of these attempt to improve the localization after a fixed number of $K$ sensing actions. The first one is an exhaustive search of all combinations of measurement locations on a quadratic $X^a \times Y^a$, $X^a = Y^a$ action grid. Each of the $\binom{X^a \cdot Y^a}{K}$ combinations is considered and the plan with the largest determinant of the Fisher information is selected.

The second path planner is myopic and considers all neighbouring positions on the action grid for the next sensing action. For each of those sensing actions, a uniform sampling of possible measurements is made and the updated belief is computed for each measurement sample. The action value of a sensing action is defined as the maximum value of the determinants of the updated beliefs. This means, the action value is the worst case of the possible determinants of the updated belief covariances. Then the sensing action is selected that minimizes the action value. This amounts to a worst-case planning.

The third path planner in [Tokekar et al., 2011] extends the myopic algorithm to a min-max tree. For each neighbouring position on the action grid, a top-level *action node* is created. An action node has multiple *bearing nodes* as child nodes, one for each sample in a uniform sampling of possible measurements. For each of those bearing nodes, the updated belief is computed. The bearing nodes again have action nodes as children until the corresponding tree reaches a depth of $K$. The value of a leaf node is defined as the determinant of the belief covariance. For each bearing node, the value is the worst value of its child nodes, similar as the

worst-case planning in the myopic planner. For each action node, the value is the best value of its child nodes. Then the top-level action node with the highest value is executed.

Both, the first and the third path planner are executed as open loop policies and do not consider replanning. The first path planner is executed once and then $K$ sensing actions are executed. For the min-max tree this is more complicated. After each bearing measurement, the belief is updated and the bearing node whose belief is most similar to the updated belief is selected as the new root. The similarity is computed using the Bhattacharyya distance, a similarity measure for two probability distributions. Then, from the new root, the action node with the highest value is executed, until $K$ measurements are taken.

In [Vander Hook et al., 2015] an algorithm for one and multiple sensor platforms is presented. In both cases, the path planner assumes that the target is at the closest position on the $3\sigma$ confidence ellipsoid. This idea is visualized in Figure 4.2. In the case of a single platform, it is shown that under the assumption of the known target position, it is optimal to select two measurement locations. One of these is on the direct line between target position and current platform position. The other can be derived analytically, if the number of measurements taken at each measurement location is known. The optimization is then performed by an exhaustive search over the number of measurements at each location, as well as a numeric optimization over the distance of the first measurement. This optimization uses the Fisher information and takes travel time and measurement duration into account. The outcome that the target might not be on the closest position on the $3\sigma$ confidence ellipsoid is not considered during the planning. Instead, the algorithm replans on the updated belief, if the target is not localized after the measurements.

The path planner in [Cliff et al., 2015] is myopic and selects the sensing action that minimizes the Shannon entropy of the belief updated with the expected measurement. The required time to move to the next measurement location is ignored in this approach.

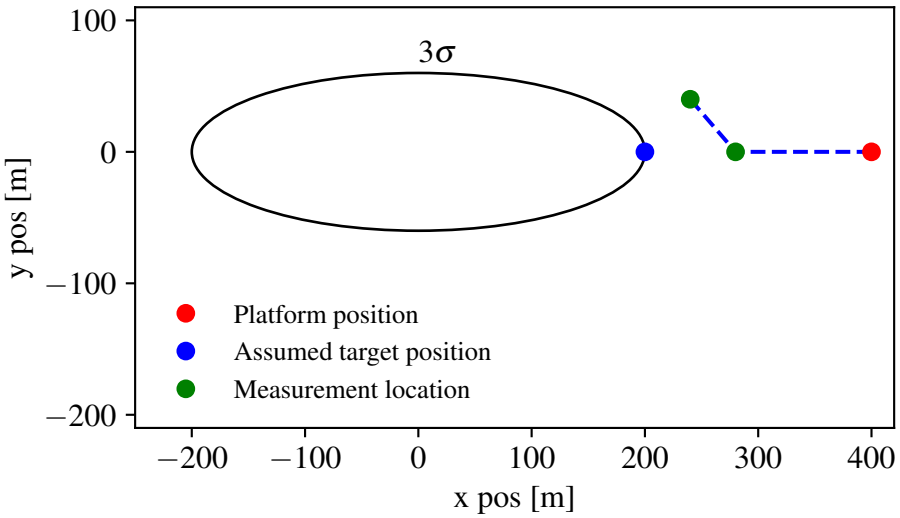



**Figure 4.2:** Visualization of the path planner from [Vander Hook et al., 2015]. The belief is represented by the $3\sigma$ confidence ellipsoid. The target is assumed to be at the position on the $3\sigma$ confidence ellipsoid that is closest to the platform.

Similarly, the path planner in [Vander Hook et al., 2014] is myopic and ignores the travel cost. It chooses the next measurement location perpendicular to the major axis of the confidence ellipsoid. The distance is determined analytically, such that there is a fixed probability that the target is on one side of the sensor. This is motivated by the characteristics of the used sensor. The sensor consists of a loop antenna, whose antenna pattern has two equal maxima. Therefore, the direction of a bearing measurement is ambiguous by 180°. To mitigate this ambiguity, the path planner chooses the measurement with a sufficient distance, such that the direction of the bearing measurement can be considered unambiguous with high probability.

[Vander Hook et al., 2014] also contains an initialization procedure for the initial belief, in which the platform moves in four directions along two orthogonal lines until the target cannot be detected any more. Then the $3\sigma$ confidence ellipsoid of the initial belief is a circle fitted on the positions where the target is not detectable any more.

### 4.1.2 Contributions of this chapter

The second and third path planner in [Tokekar et al., 2011] are the only ones that consider the stochastic outcome of future measurements, but they do not consider

the trade-off between taking a measurement and moving to a different position. As the movement between two measurements corresponds to one edge on a regular grid, these algorithms can be considered as a reduction to the case of measurements in regular time intervals with constant travel distance. This is similar to the trajectories discussed in Section 3.3.3 and does not use the capability of moving to any next measurement location. Contrary, the first path planner in [Tokekar et al., 2011] considers the fact that the platform can move to any measurement location between two measurements, but does not include the stochastic outcome of future measurements. The path planners of [Cliff et al., 2015] and [Vander Hook et al., 2014] are both myopic and also do not consider the travel time. However, both use the fact that platforms can move arbitrarily between measurements. Only the path planner in [Vander Hook et al., 2015] considers the measurement duration and travel time in its optimization, but does not consider the full uncertainty of the belief. Instead, it optimizes under the assumption of a single target position on the $3\sigma$ confidence ellipsoid. It does not consider the possibility that the target might not be at this position.

In contrast to the presented literature, this chapter describes a path planner that does not have these limitations. The path planner is based on the policy rollout principle and considers multiple possible target positions, dependent on the current belief. The outcome of future measurements during the planning depends on these different target positions. The planning is nonmyopic and considers all future decision steps until the target is localized. The cost function is based on the time required by the sensing actions, considering the measurement duration and travel time. The total cost is the time until the localization of the target is sufficiently accurate.

We will compare this rollout path planner with the algorithms from [Cliff et al., 2015] and [Vander Hook et al., 2015]. The first one is used for comparison because myopic algorithms are widely used in sensor management. The second one is used because it is the only one that explicitly considers the measurement duration in its

optimization. The chosen base policy of the policy rollout is similar to the path planner from [Vander Hook et al., 2014].

## 4.2 Problem description

In this section, a mathematical formulation of the joint state space of target and sensor system, the transition function, the belief and cost function of the emitter localization problem with a UAV is given.

### 4.2.1 State space and transition

The problem is modelled two-dimensional and the joint state

$$\mathbf{x}_k = \begin{pmatrix} \mathbf{x}^t \\ \mathbf{x}_k^s \end{pmatrix} = (x^t, y^t, x_k^s, y_k^s)^{\mathrm{T}} \in \mathcal{X} \subset \mathbb{R}^4 \tag{4.1}$$

consists of the stationary target position $\mathbf{x}^t = (x^t, y^t)^{\mathrm{T}}$, and the current position of the sensor platform $\mathbf{x}_k^s = (x_k^s, y_k^s)^{\mathrm{T}}$. The joint state space $\mathcal{X} = \mathcal{X}^t \times \mathcal{X}^s$ consists of the target state space $\mathcal{X}^t \subset \mathbb{R}^2$ and sensor state space $\mathcal{X}^s \subseteq \mathbb{R}^2$. We assume both sets to be nonempty, $\mathcal{X}^t$ to be bounded, $\mathcal{X}^s$ to be convex and $\mathcal{X}^t \subseteq \mathcal{X}^s$. The assumption that $\mathcal{X}^t$ is bounded allows to represent the probability density of the target position on a finite grid. The assumption that $\mathcal{X}^s$ is convex means that the shortest path between two measurement locations is a straight line. This assumption can be relaxed, but then the computation of the travel distance between two locations becomes more complex. The requirement that $\mathcal{X}^t \subseteq \mathcal{X}^s$ is to assure that the platform can move arbitrarily close towards the target. We use this assumption in the definition of the action space. If $\mathcal{X}^t$ and $\mathcal{X}^s$ are equal, we refer to $\mathcal{X}^t = \mathcal{X}^s$ as the *scenario area*.

We assume that the emitter is continuously active and therefore at each measurement step $k$ a measurement $z_k \in [-\pi, \pi]$ is generated. This measurement

$$z_k = \operatorname{atan2}(y^t - y_k^s, x^t - x_k^s) + w_k^m \tag{4.2}$$

$$= h_k^{\mathrm{DF}}(\mathbf{x}^t) + w_k^m \tag{4.3}$$

is based on the true bearing and additive Gaussian distributed measurement noise with $w_k^m \sim \mathcal{N}(0, \sigma^2)$ and known standard deviation $\sigma$. The measurement process takes an amount of time, denoted $t^m$. We denote this amount of time *measurement duration*.

After each measurement, a path planner chooses an action $\mathbf{a}_k \in \mathcal{A} = \mathcal{X}^s$ to move the platform to a position where the next measurement is taken. The sensor transition function is defined as

$$f^s(\mathbf{x}_k^s, \mathbf{a}_k) = \mathbf{a}_k \ . \tag{4.4}$$

The movement leads to a distance cost in time of

$$t^d(\mathbf{x}_k^s, \mathbf{a}_k) = \frac{\|\mathbf{x}_k^s - \mathbf{a}_k\|_2}{v^s} \tag{4.5}$$

where $v^s$ is the speed of the platform. We assume that the effect of acceleration is negligible and that the platform takes the direct path, however, it is possible to relax this assumption with a different function $t^d$.

Given the preceding definitions, the transition function of the state is

$$\mathbf{x}_{k+1} = f^a(\mathbf{x}_k, \mathbf{a}_k) = \begin{pmatrix} \mathbf{x}^t \\ \mathbf{a}_k \end{pmatrix} \tag{4.6}$$

with an associated cost in time of

$$c(\mathbf{x}_k^s, \mathbf{a}_k) = t^d(\mathbf{x}_k^s, \mathbf{a}_k) + t^m \ . \tag{4.7}$$

This cost is received each time the platform moves and takes a measurement, thereby incentivizing it to localize the target as fast as possible.

### 4.2.2 Belief state

The target state is not directly available to the path planner, but only indirectly observed via the received measurements. The measurements can be integrated into a belief about the target position

$$b_k^t(\mathbf{x}) = p(\mathrm{X}^t = \mathbf{x} \mid b_0^t, \mathbf{a}_0, z_1, \mathbf{a}_1, ..., \mathbf{a}_{k-1}, z_k) \ . \tag{4.8}$$

The initial belief $b_0^t$ denotes any prior knowledge about the target position or assumptions like a uniform prior distribution. The knowledge about the joint state is the belief at time k, which includes the fully observable platform position:

$$b_k(\mathbf{x}) = p(\mathrm{X} = \mathbf{x}) \tag{4.9}$$

$$= b_k^t(\mathbf{x}^t) \cdot \delta(\mathbf{x}^s - \mathbf{x}_k^s) \ . \tag{4.10}$$

Here $\delta$ corresponds to the Dirac delta function (see Sections 2.2.4 and 3.3.1). Effectively, this means that $\mathbf{x}_k^s$ is known to the path planner.

### 4.2.3 Optimization objective

The sensing task considered in this thesis is to localize the target as fast as possible. The corresponding metric is the *time until localization*. This is defined as the time until the accuracy of the localization is sufficient. The accuracy of the localization is given by the expected RMSE $\mu : \mathcal{B} \to \mathbb{R}$ of the belief, given by (3.57). The target is considered to be localized, when the expected RMSE of the belief is below a *localization accuracy threshold* $\overline{\mu}$. Formally, this means the POMDP terminates at decision step K, when belief $b_\mathrm{K}$ is in the set of termination beliefs

$$\mathcal{T}^{\mathcal{B}} = \{b \in \mathcal{B} : \mu(b) < \overline{\mu}\} \ . \tag{4.11}$$

The objective is to find a path planner that minimizes the expected time until localization, which means a policy

$$\pi^* = \underset{\pi}{\operatorname{argmin}}\, \mathbb{E}\left[\sum_{k=0}^{\mathrm{K}-1} c(\mathrm{X}_k^s, \pi(\mathrm{B}_k))\right] . \tag{4.12}$$

As defined in Section 2.2.1, the random variable K indicates the first decision step when the belief $\mathrm{B}_k$ is in the set $\mathcal{T}^B$. K depends on the target position and the encountered measurements. The sum in (4.12) should be read as: the sum of all costs until the target is localized.

When evaluating (4.12), the unknown target state $\mathrm{X}^t \sim b_0$ is distributed according to the initial belief. $\mathrm{B}_0 = b_0$ is given by the initial belief and future beliefs

$$\mathrm{B}_{k+1} = \mathrm{update}(\mathrm{B}_k, Z_k, \mathrm{X}_k^s) \tag{4.13}$$

are the result of the measurements

$$\mathrm{Z}_k = h_k^{\mathrm{DF}}(\mathrm{X}^t) + \mathrm{W}_k^m \tag{4.14}$$

where $h_k^{\mathrm{DF}}$ is described in (4.3) and $\mathrm{W}_k^m \sim \mathcal{N}(0, \sigma^2)$. The platform position is a random variable $\mathrm{X}^s$ whose value in the first decision step is equal to the initial platform position $\mathbf{x}_0^s$ and the result

$$\mathrm{X}_k^s = \pi(\mathrm{B}_k) \tag{4.15}$$

of the decisions by the path planner afterwards.

Conditioned on the target position $\mathrm{X}^t$ and the measurement noises $\mathrm{W}_k^m$, the values of all random variables are known for a given policy $\pi$. The expected value in (4.12) can therefore be approximated by Monte Carlo sampling of $\mathrm{X}^t$ and $\mathrm{W}_k^m$.

## 4.3 Path planning algorithm

The path planner is based on the policy rollout principle described in Section 2.2 and called *rollout path planner*. This section describes in detail the used base policy, the localizer, as well as the way the estimates of the action values are computed.

### 4.3.1 Localizer

We use a grid-based Bayes filter as described in Section 3.1 to compute the belief $b_k^t$ about the target position. This is computationally more demanding than for example an extended Kalman filter, but allows to capture the non-linearity in the estimation process better.

An important question is the resolution of the grid. A fixed grid at a low resolution does not allow for an accurate representation of the uncertainty. On the other hand, a fixed grid with a high resolution leads to a high amount of computational load. The rollout path planner will update the belief many times by simulated future measurements. Therefore, an inefficient implementation would lead to higher computation and planning times.

A major waste with a high resolution grid is the evaluation of the likelihood of unlikely target positions. We solve this problem by making the grid resolution and extension adaptive, and limit the grid on the area of *possible target positions*, with increasing resolution as the localization becomes more accurate. The possible target positions are computed under the assumption that the measurement noise is limited by $4\sigma$. The possible target positions are then given by the convex hull $\mathcal{C}_k^t$, which is the convex hull of the intersection of all $4\sigma$ cones and the target state space. If the target state space is convex, the convex hull is identical to the intersection. Under the assumption of bounded measurement noise, the target must be in this set. This convex hull is visualized in Figure 4.3.

The extension $\underline{x}_k^b, \overline{x}_k^b, \underline{y}_k^b, \overline{y}_k^b$ of the grid-based Bayes filter is then set to the smallest rectangle containing the possible target positions. The initial extension is set to the minimum and maximum coordinates of the target state space with

$$\underline{x}_0^b = \min_x \{(x,y) \in \mathcal{X}^t\} \tag{4.16}$$

$$\overline{x}_0^b = \max_x \{(x,y) \in \mathcal{X}^t\} \tag{4.17}$$

and $\underline{y}_0^b, \overline{y}_0^b$ analogously. The probability density $p_{kij}$ in cells without intersection with $\mathcal{X}^t$ is set to zero and to a uniform value in the remaining cells. The grid is initially created with 100 cells on the longer axis, and a proportionally smaller integer number on the other axis, which is chosen to make the grid cells closest to a square. If due to the resizing step the number of grid cells in the larger dimension becomes less than 40, the resolution in both dimensions is doubled.

An alternative to using this convex hull would be to test whether a grid cell is numerically zero and reduce the grid by those columns and rows that are zero everywhere. There are two motivations of using the approach with the convex hull instead. First, it clearly states which errors are assumed to happen. Mathematically, no grid cell would be zero because the normal distribution has positive probability density for each value. Therefore, the extension of the grid would depend on the numerical resolution of the computing platform. Second, the convex hull provides a way to detect whether outliers happened. If the $4\sigma$ cones do not intersect, this indicates that at least one of the measurements has an error higher than $4\sigma$. In this chapter, outliers do not occur as the simulations focus on the sensor path planning problem and outliers $> 3\sigma$ are resampled. However, in the experiments in Chapter 6 the path planning would then be terminated and the last estimate returned.

If an initial belief about the target position is available, the grid is initialized using this probability density. The point estimate of the target position is based on the maximum a posteriori estimate. However, when the initial belief is uniform,

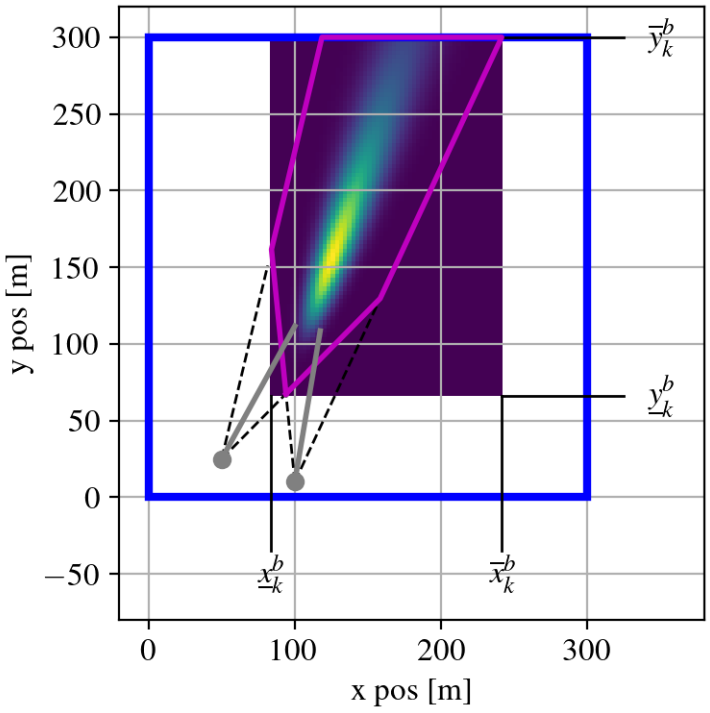



**Figure 4.3:** Extension of the grid-based Bayes filter. The dashed black lines indicate the $\pm 4\sigma$ cones of two measurements (gray). Under the assumption that the measurement noise is bounded by $4\sigma$ and given the target state space $\mathcal{X}^t$ (scenario area, blue), we compute the possible target positions $\mathcal{C}_2^t$ (magenta). The extension of the grid-based Bayes filter is then reduced the smallest rectangle covering $\mathcal{C}_2^t$.

the probability density has no clearly identified peak until two measurements are available. In this case the point estimate is set to

$$\tilde{\mathbf{x}}_k^t = \begin{cases} \tilde{\mathbf{x}}_k^{t,ev} & \text{if } k = 1 \\ \tilde{\mathbf{x}}_k^{t,map} & \text{if } k > 1 \end{cases} \tag{4.18}$$

where the expected value $\tilde{\mathbf{x}}^{t,ev}$ and maximum a-posteriori estimate $\tilde{\mathbf{x}}^{t,map}$ are given by (3.30) and (3.31), respectively. Based on this point estimate, the expected RMSE $\mu(\tilde{\mathbf{x}}_k^t, b_k^t)$ is computed according to (3.58). In addition, the covariance $\tilde{\mathbf{P}}_k^t$ of this point estimate is computed by (3.34).

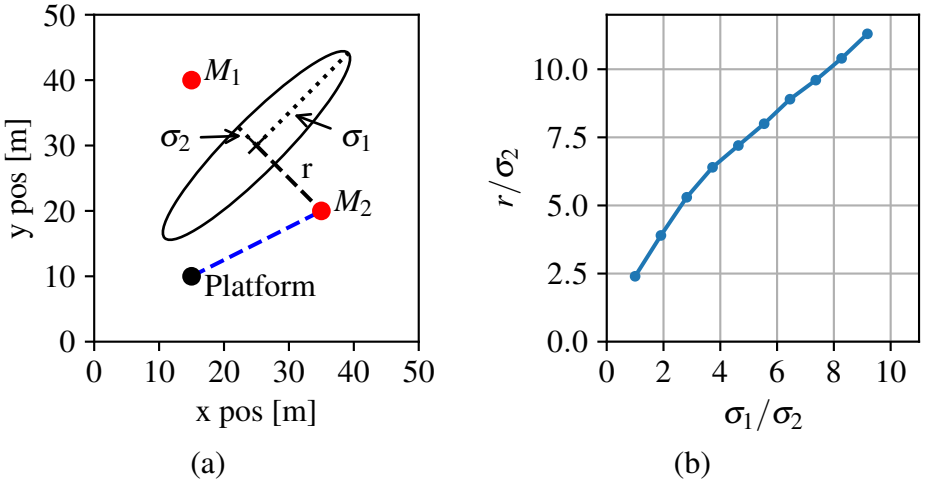



**Figure 4.4:** Base policy for the policy rollout. (a) $M_1$ and $M_2$ are the potential next measurement locations, perpendicular to the major axis of the confidence ellipsoid. From those two alternatives the one that is closer to the platform is chosen, here $M_2$. (b) The distance $r$ normalized by the smaller axis $\sigma_2$ dependent on the ratio of the standard deviations $\sigma_1/\sigma_2$ . Note that $\sigma_1/\sigma_2 \geq 1$ by definition.

### 4.3.2 Base policy

The base policy $\pi^B$ represents a heuristic solution for the problem. In our case we simplify the problem by ignoring movement costs and selecting the sensing action that leads to approximately the lowest expected RMSE after the next measurement. For this, we follow the intuition that, if the belief about the target position were Gaussian, the next measurement location should be perpendicular to the major axis of the confidence ellipsoid. This idea is similar to the path planner in [Vander Hook et al., 2014]. The main difference is that in the base policy, the distance is selected such that the expected RMSE of the updated belief is minimal, while in [Vander Hook et al., 2014] the distance was selected such that the bearing measurement is unambiguous with high probability.

Figure 4.4a visualizes this idea. The confidence ellipsoid belongs to the Gaussian approximation $(\tilde{\mathbf{x}}_k^t, \tilde{\mathbf{P}}_k^t)$ of the belief. The covariance $\tilde{\mathbf{P}}_k^t$ is computed according to (3.34). Then the next measurement location is determined by the distance $r$ to the point estimate $\tilde{\mathbf{x}}_k^t$ of the target position. There are two possible measurement locations with distance $r$, which are denoted by $M_1$ and $M_2$ in the figure.

The optimal distance $r$ between the measurement location and the point estimate of the target position is dependent on the ratio between semi-major and semi-minor axis of the confidence ellipsoid. The distance $r$ should increase if the semi-major axis $\sigma_1$ becomes larger in relation to the semi-minor axis $\sigma_2$, to increase the probability that the measurement is close to perpendicular to $\sigma_1$. On the other hand, a bearing measurement contains less information about the target position if taken from further away, as the likelihood of the measurement spreads over a larger region. This can be seen directly by the formula of the Fisher information (3.80).

While the Fisher information increases when $r$ decreases, the optimal distance is not necessarily zero. This is because the target might not be at exactly the position of the estimate point anywhere in the uncertainty region. For example, a measurement close to the estimated target position might reduce the uncertainty by a smaller amount than a measurement further away if the true target position is not at the point estimate. Figure 4.5 visualizes this possibility. The $r$ chosen by the base policy is the one which on average over the possible target positions and measurements leads to the largest reduction in uncertainty.

The optimal distance $r$ was computed offline by setting the semi-minor axis of a Gaussian shaped belief to $\sigma_2 = 1$ and varying the semi-major axis. For a given length of the semi-major axis, the expected RMSE of the updated belief was computed for each distance $r$, and the distance with the lowest expected RMSE was selected. The expected RMSE of the updated belief was determined by computing the updated belief for all possible measurements based on the possible target positions and the measurement noise. For efficiency reasons the possible measurements were binned into $1^\circ$ bins. Figure 4.4b shows the resulting optimal distance $r$ normalized by $\sigma_2$, dependent on the ratio of the axes. Values higher than precomputed are linearly interpolated from the last values.

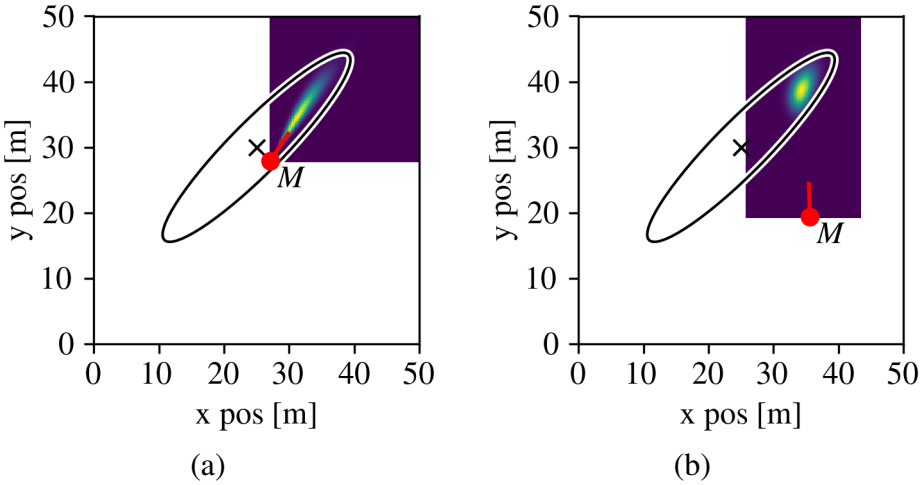



**Figure 4.5:** Visualization that the best measurement location is not necessarily the one closest to the current estimate. The measurement is shown in red, the heat map represents the updated belief. (a) An update at 3 m distance leads to an expected RMSE of 3.69 m. (b) An update at 15 m distance leads to an expected RMSE of 2.09 m.

At runtime, the base policy computes the length of the axes from the Gaussian approximation (3.34) of the grid-based density. Then, based on their ratio, a piecewise linear interpolation

$$r = \sigma_2 \cdot \mathrm{PLI}\left[\frac{\sigma_1}{\sigma_2}\right] \tag{4.19}$$

of the precomputed values is performed. As can be seen in Figure 4.4a, there are two possible candidate measurement locations $M_1$ and $M_2$ with range $r$. The base policy selects as sensing action the one that is nearer to the platform, which is $M_2$ in the figure.

### 4.3.3 Policy rollout

The path planner is based on the policy rollout principle, described in Section 2.2. This class of algorithms approximates future actions with a base policy. The action value of $\mathbf{a}$ is given by the sum of the immediate cost $c(\mathbf{x}_k^s, \mathbf{a})$ and the expected value of the future costs $v^B(b_{k+1})$ when following the base policy. We approximate the expected value by drawing samples of the target state and the measurements during the rollout. The rollout is executed until the expected RMSE is below the

localization accuracy threshold. This corresponds to the termination criterion of the stated problem. Therefore, the rollout path planner has a planning horizon of the full problem length. The sampled action value is

$$\mathrm{Q}^B(b_k, \mathbf{a}) = c(\mathbf{x}_k^s, \mathbf{a}) + \frac{1}{R} \sum_{j=1}^{R} \cdot \sum_{i=k+1}^{\mathrm{K}_{k\mathbf{a}j}-1} c(\mathrm{X}_{k\mathbf{a}ji}^s, \pi^B(B_{k\mathbf{a}ji})) \tag{4.20}$$

where $R$ is the number of rollouts per action. It should be noted that for different rollouts $j$, the maximal index $\mathrm{K}_{k\mathbf{a}j}$ can be different.

The random variables in the policy rollout follow the same transitions as the problem described in Section 4.2.3. The main difference is that instead of a single random sequence indexed by the decision step $k$, here multiple random sequences are simulated to evaluate different futures. The random variables are indexed by $k\mathbf{a}ji$, corresponding to the $i$-th future decision step in the $j$-th rollout for action $\mathbf{a}$ during the planning at decision step $k$. The index $k$ corresponds to the current decision step, while the index $i$ refers to future decision steps that are simulated in the rollout. The platform position at decision step $k+1$ is determined by the evaluated action $\mathbf{a}$ with $\mathrm{X}_{k\mathbf{a}j[k+1]}^s = \mathbf{a}$. Future platform positions are determined by $\mathrm{X}_{k\mathbf{a}j[i+1]}^s = \pi^B(\mathrm{B}_{k\mathbf{a}ji})$, similar to (4.15). At decision step $k$, the begin of the rollout, the current belief is used to sample the target position for the rollout, with $\mathrm{X}_{k\mathbf{a}j}^t \sim b_k$. The rollout belief $\mathrm{B}_{k\mathbf{a}jk} = b_k$ is identical to the current belief at the start of the rollout and updated with simulated measurements afterwards.

With

$$\hat{q}^B(b_k, \mathbf{a}, \Omega_{kj}) = \sum_{i=k+1}^{\mathrm{K}_{k\mathbf{a}j}-1} c(\mathrm{X}_{k\mathbf{a}ji}^s, \pi^B(\mathrm{B}_{k\mathbf{a}ji})) \tag{4.21}$$

we denote the sampled action value for action $\mathbf{a}$, belief $b_k$, and samples $\Omega_{kj}$. The computation of function (4.21) is called a *rollout* and computed by Algorithm 1. This algorithm receives as input the current sensor position $\mathbf{x}_k^s$, the current belief about the target position $b_k^t$ and the action $\mathbf{a}$ that should be evaluated. The realization $\omega_{kj} = (\omega_{kj}^x, \omega_{kj}^y, \omega_{kj}^m)$ of $\Omega_{kj}$ is represented with the x- and y-position of the sampled target position $(\omega_{kj}^x, \omega_{kj}^y)$, and a parameter $\omega_{kj}^m$ that describes how the measurements

should be computed. This can be either a random seed for the measurement noise sequence or a symbol 'None', meaning that the measurement noise should be zero.

**Algorithm 1** Rollout

**procedure** ROLLOUT($b^t, \mathbf{x}^s, \mathbf{a}, \omega^x, \omega^y, \omega^m$)
  $s \leftarrow 0$ ▷ Sum of cost
  **while** $\mu(b^t) \geq \overline{\mu}$ **do**
    $s \leftarrow s + c(\mathbf{x}^s, \mathbf{a})$
    $\mathbf{x}^s \leftarrow \mathbf{a}$
    **if** $\omega^m =$ 'None' **then**
      $w^m \leftarrow 0$
    **else**
      $w^m \sim \omega^m$ ▷ Random sequence with seed $\omega^m$
    $x^s, y^s \leftarrow \mathbf{x}^s$ ▷ Extract vector components
    $z \leftarrow \text{ATAN2}(\omega^x - x^s, \omega^y - y^s) + w^m$
    $b^t \leftarrow \text{UPDATE}(b^t, z, \mathbf{x}^s)$
    $\mathbf{a} \leftarrow \pi^B(b^t, \mathbf{x}^s)$
  **return** $s$

As a shorthand notation we define

$$\hat{q}^B(b_k, \mathbf{a}, \Omega_{k[1:R]}) := \frac{1}{R} \sum_{j=1}^{R} \hat{q}^B(b_k, \mathbf{a}, \Omega_{kj}) \tag{4.22}$$

which is an convenient way to express the execution of $R$ rollouts.

The path planner presented in this chapter uses deterministic samples for the target position and a fixed measurement noise of zero. The motivation to use a deterministic instead of a Monte Carlo approach is that deterministic samples can represent the belief in a structured way and therefore allow for more efficient computation of the expected value. A comparison between deterministic samples and Monte Carlo samples is made in Chapter 5.

The sampling mechanism to determine the target position samples $\mathrm{X}^t_{k\mathbf{a}j}$ is based on the algorithm [Klumpp and Hanebeck, 2008]. This algorithm works by subdividing the probability density of the target position into areas of equal probability mass. Therefore, the samples represent the probability density well even with a small amount of samples. Figure 4.6 shows an example of those deterministic samples. In each iteration, the algorithm has a given set of samples where each rep-

resents a section of the outcome space. Then each sample is split into two samples which represent the probability density on this section better. As an effect of this regular splitting of the probability density, the samples are best distributed if their number is a power of two.

The future measurements are approximated without noise, which means

$$\mathrm{W}^{m}_{k\mathbf{a}ji} = 0 \,. \tag{4.23}$$

Drawing deterministic samples from the measurement noise would impose a great computational demand, as the measurement noise is assumed to be independent and identically distributed. When computing a fixed number of deterministic samples for each measurement, the combinations would exponentially increase with the number of future measurements. We therefore assume that the main source of uncertainty lies in the belief about the target position. The samples $\Omega_{kj}$ have therefore the measurement parameter $\omega^m$ set to 'None', when passed to Algorithm 1. The option of sampling the measurement noise randomly is used in Chapter 5.

### 4.3.4 Search for the optimal action

The previous sections described how the sampled action value $\mathrm{Q}^{B}(b_k, \mathbf{a})$ of a specific action $\mathbf{a}$ can be computed. To complete the rollout path planner, we further need to define the action space and a procedure to find the optimal action.

In principle, one could evaluate all possible next platform positions in the action space $\mathcal{A} = \mathcal{X}^s$. However, the sensor state space could be very large, potentially even unbounded, making an uninformed search for the optimal action difficult. Instead, we use additional knowledge about the problem to limit the action space. We know from (3.80) that the information content of a bearing measurement decreases as the distance between sensor and target increases. Therefore, we can state the rule that measurement locations should not be too far from the possible target positions.

We define those measurement locations as

$$\mathcal{X}^{m}_{k} = \left[\underline{x}^{b}_{k} - \delta x^{b}_{k}, \overline{x}^{b}_{k} + \delta x^{b}_{k}\right] \times \left[\underline{y}^{b}_{k} - \delta y^{b}_{k}, \overline{y}^{b}_{k} + \delta y^{b}_{k}\right] \subset \mathbb{R}^2 \,, \tag{4.24}$$

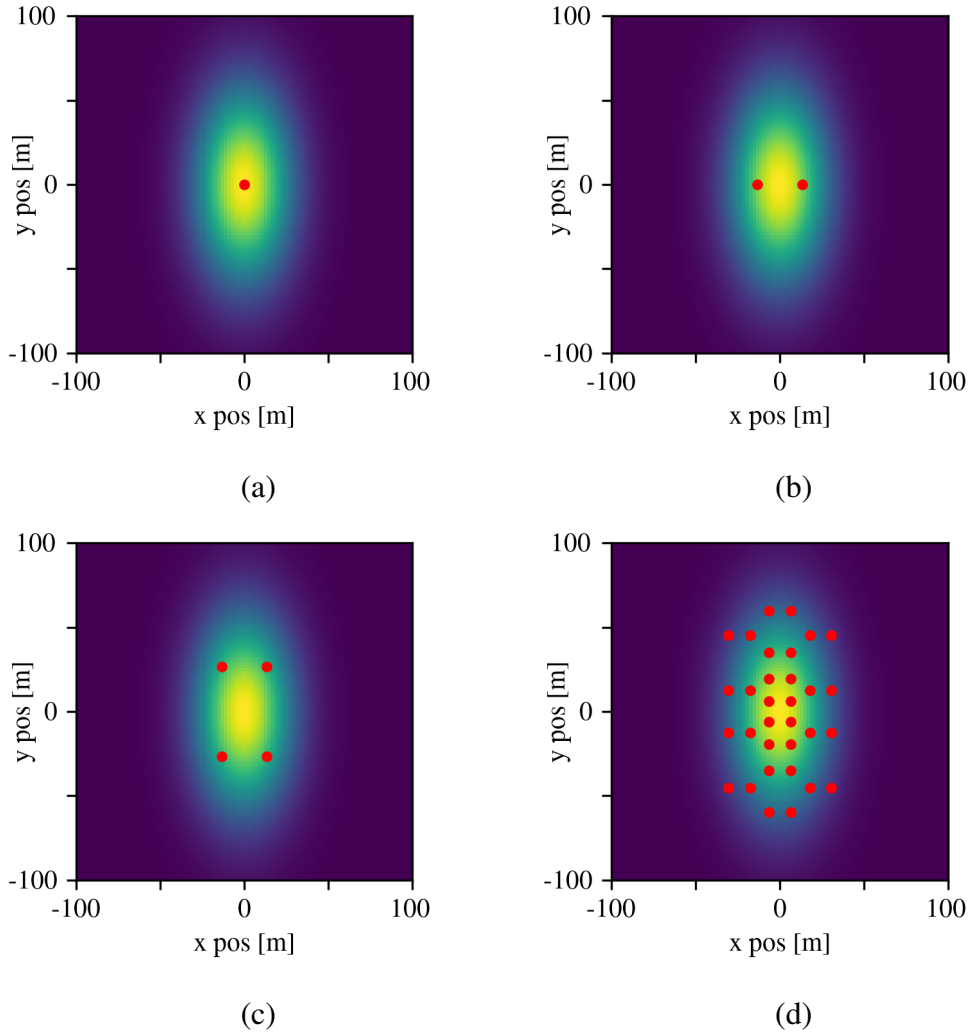



**Figure 4.6:** Deterministic samples of a normal distribution (Covariance = diag($20^2$, $40^2$)), discretized to a 100x100 grid. (a) 1 sample, (b) 2 samples, (c) 4 samples, (d), 32 samples.

with

$$\delta x_k^b = \bar{x}_k^b - \underline{x}_k^b \tag{4.25}$$

$$\delta y_k^b = \bar{y}_k^b - \underline{y}_k^b \tag{4.26}$$

$$\tag{4.27}$$

being the width and the height of the belief grid. Effectively this means that $\mathcal{X}^m$ is the extension of the grid-based Bayes filter, tripled in width and height. Figure 4.7

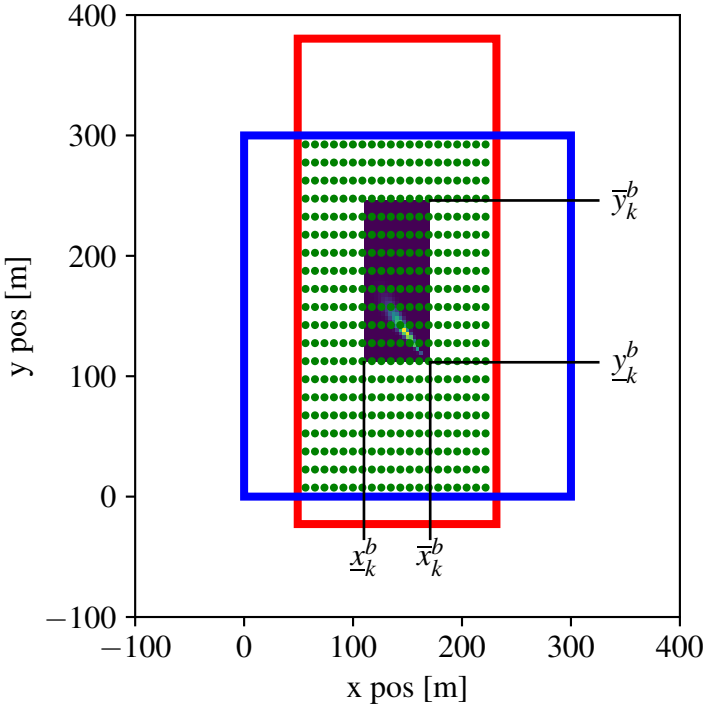



**Figure 4.7:** Visualization of the action space. The size of the belief area is extended by three in each dimension leading to $\mathcal{X}^m$, the red coloured rectangle. The intersection between this rectangle and the state space of the platform (scenario area, blue rectangle) leads to the action space $\mathcal{A}_k$. The action space is further discretized into a $20 \times 20$ action grid $\mathbb{A}_k$ (green).

visualizes this construction. The action space $\mathcal{A}_k \subseteq \mathcal{A}$ at decision step $k$ is then the intersection

$$\mathcal{A}_k = \mathcal{X}^m \cap \mathcal{X}^s \tag{4.28}$$

between the measurement locations close to the belief and the sensor state space. Effectively this means that the action space is focused on measurement locations close to the belief grid that the platform can move to.

The search for the optimal action is performed by discretizing the action space. The discretization of $\mathcal{A}_k$ on an $X^a \times Y^a$ grid $\mathbb{A}_k$ is called the *action grid*. Then we replace the search in the continuous action space

$$\mathbf{a}_k = \underset{\mathbf{a} \in \mathcal{A}_k}{\operatorname{argmin}}\ \hat{q}^B(b_k, \mathbf{a}, \omega_{k[1:R]}) \tag{4.29}$$

by an exhaustive evaluation

$$\mathbf{a}_k = \underset{\mathbf{a}\in\mathbb{A}_k}{\operatorname{argmin}}\ \hat{q}^B(b_k, \mathbf{a}, \omega_{k[1:R]})\ . \tag{4.30}$$

of the action grid, where $\omega_{k[1:R]}$ consists of the deterministic samples $\omega_{kj}^x, \omega_{kj}^y$, and $\omega_{kj}^m =$ 'None' for $1 \leq j \leq R$.

It should be noted that the rollout path planner is constrained to the action space, but the base policy selects its actions unconstrained in $\mathbb{R}^2$, potentially even outside of the scenario area. This is a valid option for the scenarios used in this and the following chapter because the scenario area is used here primarily for two reasons: to limit the area searched for the optimal action and to make the target state space bounded, allowing for a grid-based Bayes filter as localizer. Even though the base policy leaves the scenario area, the resulting rollout path planner still performs well. The rollout path planner will also only select actions inside the scenario area.

## 4.4 Evaluation

In this section the previously described rollout path planner is evaluated. Based on two scenarios, the algorithm is compared with two path planners from the literature, an extension of a path planner from the literature, and the base policy.

### 4.4.1 Scenarios

Two different scenarios are used for the evaluation. In the first scenario, the target is positioned at different distances from the initial platform position. We assume that the scenario area is known to the localizer, otherwise no additional information is provided. The localizer therefore assumes a uniform initial belief of the target position. The setup of this scenario is displayed in Figure 4.8a. It consists of a $300\,\mathrm{m} \times 300\,\mathrm{m}$ area. Before the execution of each path planner, a measurement is taken to initialize the localizer. This means the first execution of the path planner is performed on the belief $b_1$, with the sensing action $\mathbf{a}_0 = (150, 0)^\mathrm{T}$ being fixed. The different possible target positions are on a line at $x^t = 150\,\mathrm{m}$, between $y_0^t = 25\,\mathrm{m}$ and $y_{10}^t = 275\,\mathrm{m}$, spaced in 25 m intervals. The index of the $y$-coordinates represents an

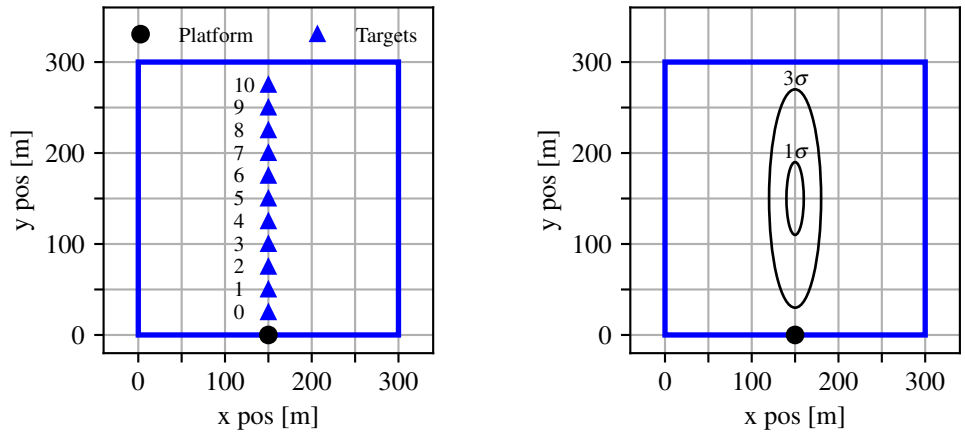
index of the target position, increasing in distance. The target positions are chosen to facilitate a qualitative analysis of different distances, but are not an unbiased sample of the target state space. For example there are more positions in the target state space with a distance of 150 m to the initial platform position, than with 25 m.

Therefore, the first scenario is not suitable to determine an average performance of a path planner, as the considered target positions are not representative of a uniform distribution over the target state space. Instead, they are intended to understand the behaviour of the algorithms better. In contrast, the second scenario is intended to determine the average performance of the path planners when the belief corresponds to the true uncertainty about the target position. A distribution about possible target positions is fixed, and multiple Monte Carlo simulations are performed. In each simulation, the true target position for this simulation run will be drawn once from this distribution. The distribution is known to the localizer as initial belief $b_0$.

The setup for Scenario 2 is shown in Figure 4.8b. The distribution of the target position and initial belief $b_0$ is a bivariate Gaussian distribution with standard deviation in $x$ of 10 m and in $y$ of 40 m, centred around the point $x = 150\,\text{m}$ and $y = 150\,\text{m}$. Target positions, which are not in the $3\sigma$ confidence ellipsoid of the initial belief, are discarded and resampled. This was done to adhere to the assumption of [Vander Hook et al., 2015] that the closest possible target position is on the $3\sigma$ confidence ellipsoid. The first execution of the path planner is performed on the initial belief $b_0$, without any update from a measurement.

For each scenario, we consider three different measurement durations. We do not vary the platform speed, as only its relative value to the measurement duration is important. For example, a parameter set with halved measurement duration and doubled platform speed would simply lead to a halving of the time until localization. If one trajectory is better than another in the original parameter set, it would be in the second parameter set as well.

We choose the value for the longest measurement duration similar to the value presented in [Cliff et al., 2015] and smaller values for faster rotating platforms.




(a) (b)

**Figure 4.8:** Visualization of the scenarios. (a) Scenario 1: The true target positions are indexed with increasing distance. (b) Scenario 2: The true target position of each Monte Carlo run is randomly drawn from a bivariate normal distribution.

**Table 4.1:** Parameters of the scenarios

| Configuration | Parameter | Value | Description |
|---|---|---|---|
| Fast | $t^m$ | 2 s | Measurement duration |
| Medium | $t^m$ | 10 s | Measurement duration |
| Slow | $t^m$ | 40 s | Measurement duration |
| All | $v^s$ | 5 m/s | Platform speed |
| | $\sigma$ | 4° | Measurement standard deviation |
| | $\overline{\mu}$ | 5 m | Localization accuracy threshold |

We assume a measurement noise with standard deviation $\sigma = 4°$ and require a localization accuracy threshold of $\overline{\mu} = 5\,\text{m}$. The complete parameter set is given in Table 4.1. In both scenarios the simulations for different path planners or planner configurations are based on the same initial seeds of the random number generator. Therefore, the evaluated target positions in Scenario 2 are identical.

### 4.4.2 Simulation results

We compare the rollout path planner to the base policy, an implementation of the path planner in [Vander Hook et al., 2015], an entropy-based path planner based on the expected measurement as in [Cliff et al., 2015], and a stochastic extension of this entropy path planner, using the same deterministic sampling mechanism as in the

rollout implementation. Our implementation of [Cliff et al., 2015] does not resize the grid of the updated belief during the planning step. This is because the Shannon entropy changes with the number of grid cells and the updated beliefs would not be comparable. A small difference in our implementation to [Cliff et al., 2015] is that in the original work the expected measurement was based on the expected value of the target position, while in our implementation the expected measurement is based on the point estimate of the target position. This point estimate can be either the maximum a posteriori estimate or the expected value, as described in (4.18).

We denote the path planner in [Vander Hook et al., 2015] by *sequential optimal localization of pseudo targets* (SOPT), based on the idea of the path planner in [Vander Hook et al., 2015]: it assumes a target at the borders of the uncertainty and performs an optimal localization step for this target. We refer to the path planner in [Cliff et al., 2015] as *entropy-based path planner* (ENTPP).

The stochastic extension of ENTPP is called ENTPP-8, where 8 is the number of samples taken to compute the expected entropy. It selects the next sensing action as

$$\mathrm{A}_k = \underset{\mathbf{a}\in\mathbb{A}}{\operatorname{argmin}}\ \mathbb{E}\left[\mathcal{H}^S(\mathrm{update}(b_k, \mathrm{Z}_{k+1}, \mathbf{a}))\right] \tag{4.31}$$

where $\mathrm{Z}_{k+1}$ is determined by the measurement equation (4.14) with $\mathrm{X}^s_{k+1} = \mathbf{a}$. The expected value is evaluated in a deterministic way as above, with $\mathrm{W}^m_{k+1} = 0$ and $R$ different deterministic samples of $\mathrm{X}^t$. The ENTPP method instead uses only one sample $\mathrm{X}^t = \tilde{\mathbf{x}}^t_k$, based on the point estimate of the localizer.

All path planners use the same implementation of the grid-based Bayes filter as localizer. The sensing actions for ENTPP are evaluated on an action grid, similar to the rollout path planner. This action grid uses the same action space $\mathcal{A}_k$ (4.28), but has a higher resolution since the evaluation of an action is less costly. We use a $60 \times 60$ action grid for ENTPP and a $40 \times 40$ grid for ENTPP-8. The rollout path planner also uses 8 samples, on a $20 \times 20$ action grid.

The mean time until localization in Scenario 1, medium configuration, is illustrated in Figure 4.9. Exemplary trajectories are shown in Figure 4.10a. It can be observed that the required time of SOPT linearly increases if the target is fur-

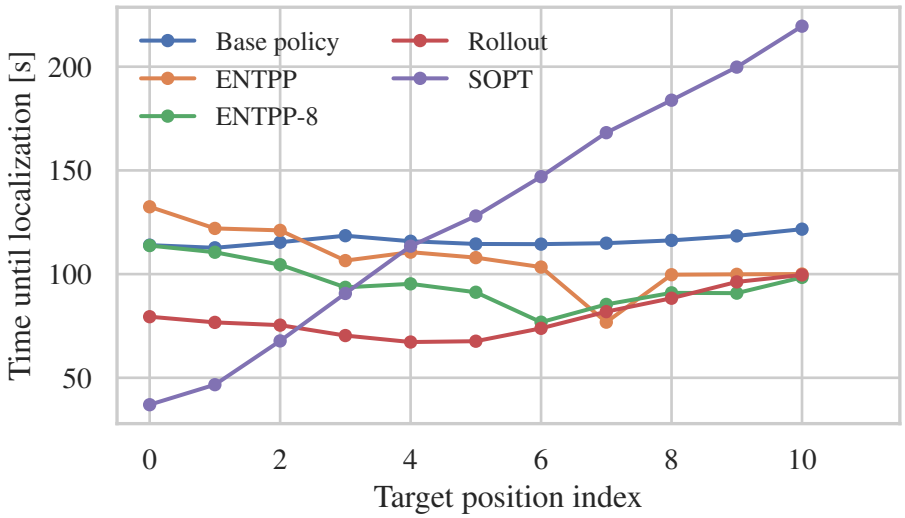
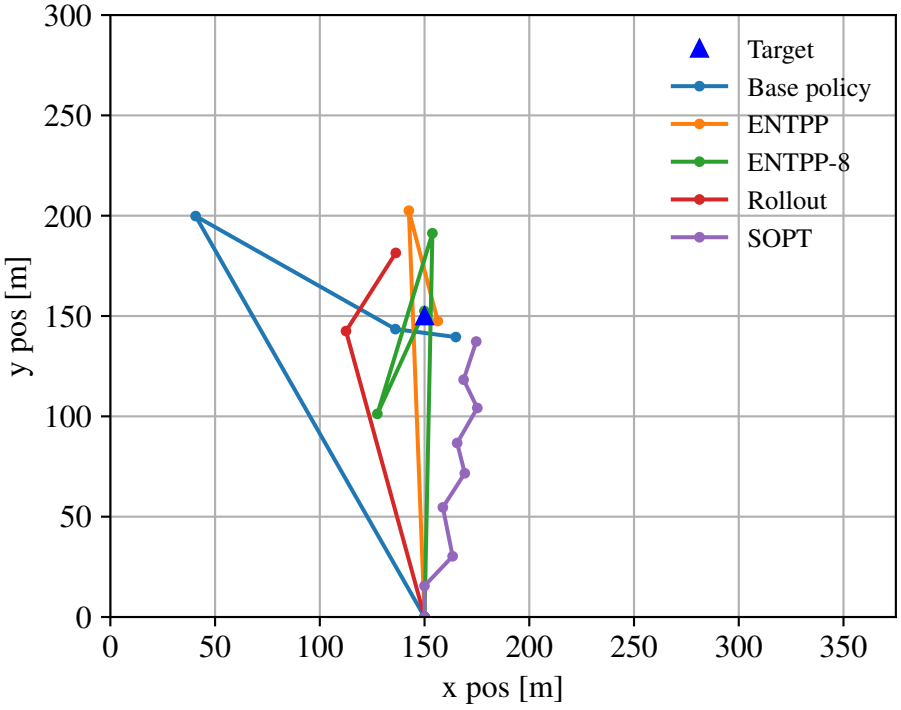


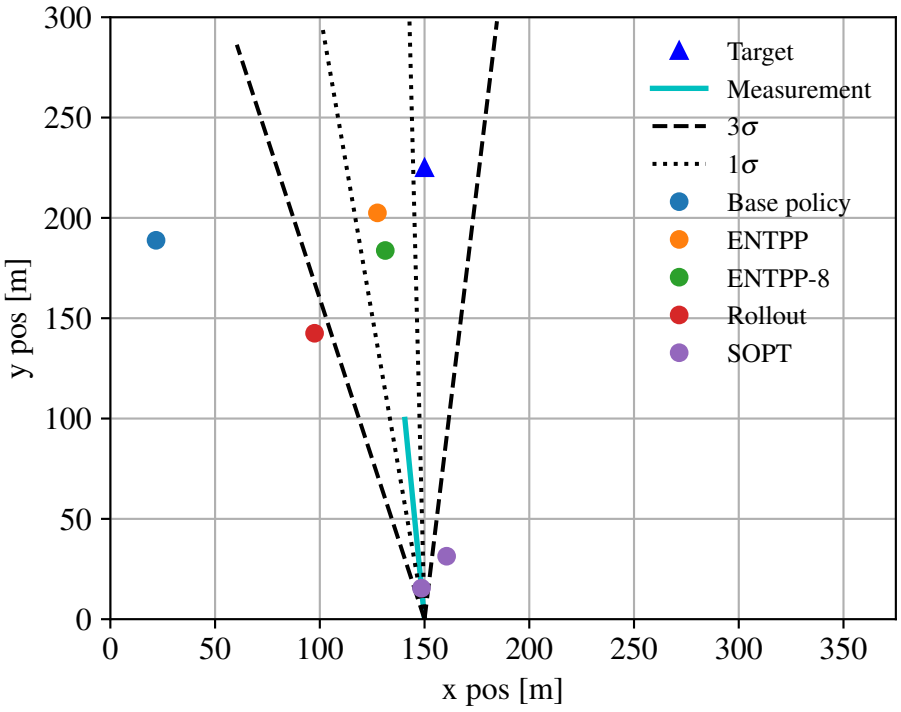

**Figure 4.9:** Comparison of the time until localization in Scenario 1, medium configuration, averaged over all Monte Carlo runs and shown for each target position index.

ther away. This is a result of its measurement location selection method, which iteratively optimizes on a close pseudo target, and therefore needs longer to reach a far-away target. Figure 4.10b shows the first sensing action $\mathbf{a}_0$ for each method, relative to the initial uncertainty. Note that SOPT determines always two measurement locations, which means $\mathbf{a}_1$ is already determined as well. The figure shows that the entropy-based path planners are centring on the uncertainty, which leads to good results when the target is actually there (target index 6 for ENTPP-8 and 7 for ENTPP). The rollout path planner instead selects a closer measurement location, farther away from the uncertainty region, which for most target ranges leads to a shorter time until localization. Figure 4.11 illustrates the adaption of the rollout path planner to the measurement duration, as it requires a smaller number of measurements in case the measurement duration is higher and vice versa.

If the measurement duration $t^m$ is short, SOPT shows strong results, as can be seen from Figure 4.12 for the fast configuration. Here, the difference for distant targets between SOPT and the other path planners is less pronounced. For close targets, it is the most effective method. However, as discussed before, given a uniform distribution over the target state space, close targets are less likely than targets further away. This is the reason, ENTPP, ENTPP-8, the rollout path planner, as




(a)




(b)

**Figure 4.10:** Behaviour of different path planners. (a) Exemplary trajectories to localize target 5 in Scenario 1, medium configuration. (b) First sensing action for each planner in Scenario 1, medium configuration to localize target 8. Note that SOPT always selects two measurement locations as a single sensing action.

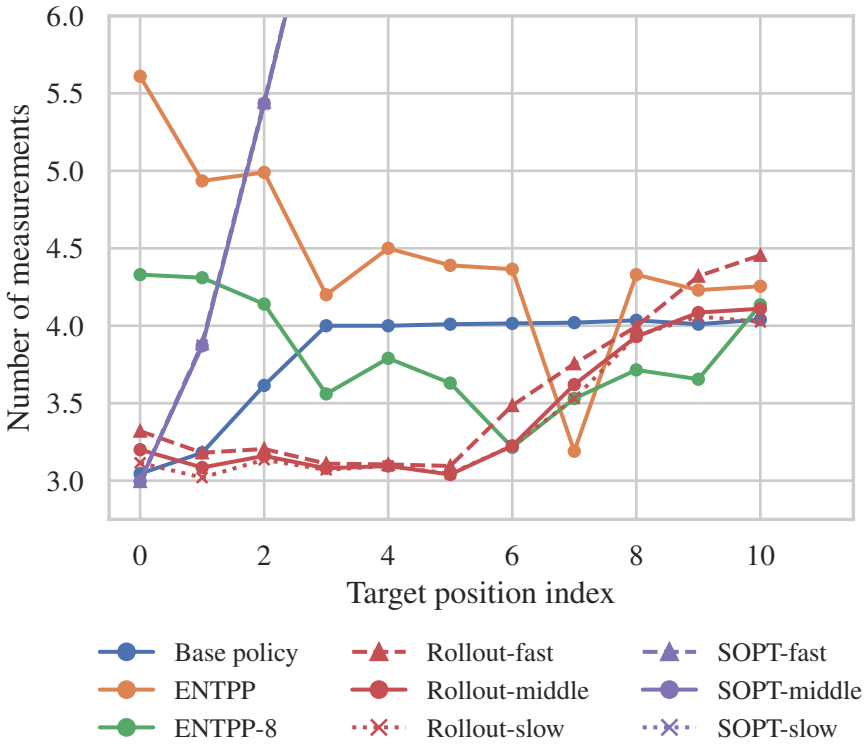



**Figure 4.11:** Comparison of the average required number of measurements, in Scenario 1, medium configuration, averaged over all Monte Carlo runs and shown for each target position index. The base policy, ENTPP, and ENTPP-8 do not vary for the different configurations, therefore only a single line is shown. SOPT would increase approximately linearly until $\approx 16$ at target position index 10, however also without visible difference on the configurations.

well as due to the Gaussian matching also the base policy, focus on the farther away targets. The order of the other path planners is similar to the medium configuration, with the rollout path planner still being good for most target ranges.

The results for the slow configuration can be found in Figure 4.13. Here SOPT shows significantly worse results than the other methods. The relative ordering of the other path planners stays similar as before, with the rollout path planner being a well performing algorithm.

Average results for Scenario 2 can be found in Table 4.2. In addition to the mean time until localization, the 95% confidence interval on the mean is given,

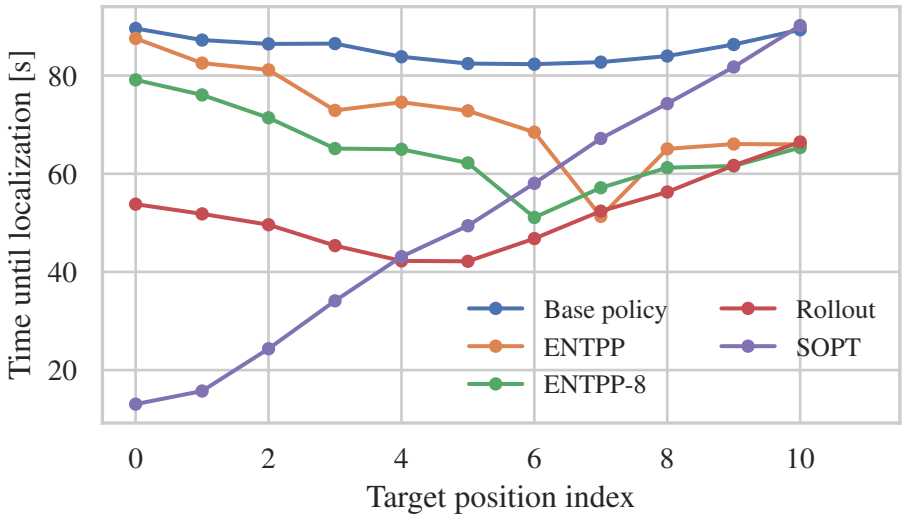



**Figure 4.12:** Comparison of the time until localization in Scenario 1, fast configuration, averaged over all Monte Carlo runs and shown for each target position index.

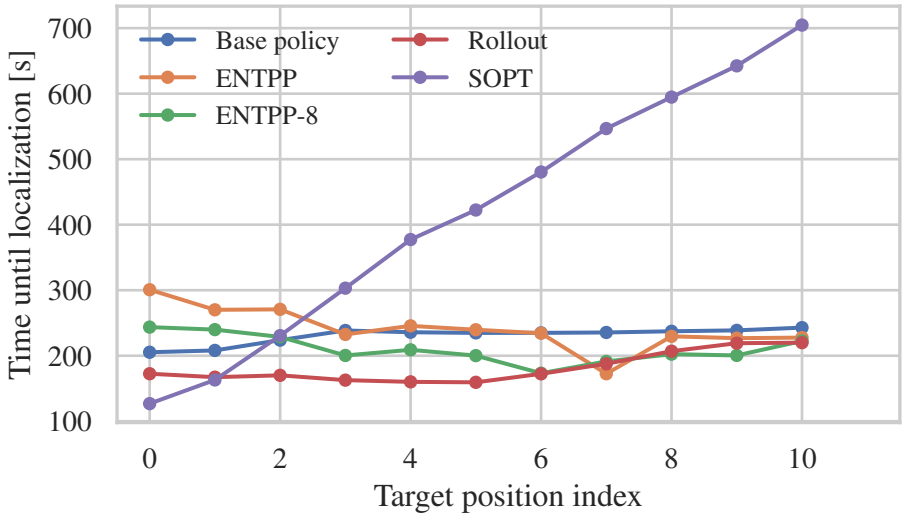



**Figure 4.13:** Comparison of the time until localization in Scenario 1, slow configuration, averaged over all Monte Carlo runs and shown for each target position index.

**Table 4.2:** Mean time until localization in seconds in Scenario 2 with 95% confidence interval.

| | Fast | Medium | Slow |
|---|---|---|---|
| Base policy | $55.03 \pm 0.30$ | $74.31 \pm 0.54$ | $146.61 \pm 1.45$ |
| ENTPP | $45.18 \pm 0.37$ | $67.25 \pm 0.71$ | $149.99 \pm 2.04$ |
| ENTPP-8 | $47.11 \pm 0.35$ | $67.35 \pm 0.63$ | $143.22 \pm 1.74$ |
| Rollout | $38.96 \pm 0.36$ | $58.22 \pm 0.58$ | $125.50 \pm 1.34$ |
| SOPT | $48.88 \pm 0.82$ | $121.14 \pm 1.98$ | $392.01 \pm 6.36$ |

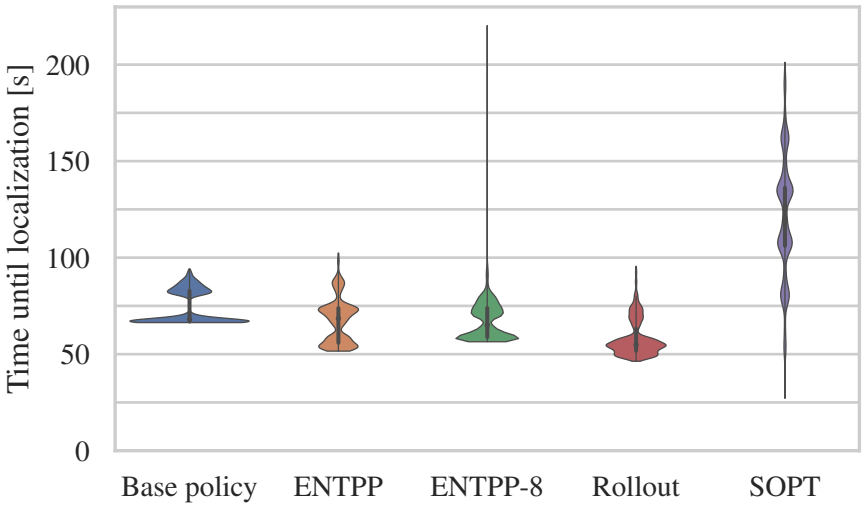



**Figure 4.14:** Comparison of the time until localization in Scenario 2 for the medium configuration. The width of the plot elements shows the frequency of a fixed time required until localization. The discontinuities correspond to different numbers of measurements.

which is computed assuming a normal distribution of the results. This is only an approximation, as can be seen in Figure 4.14, which shows that the distribution is clearly multi-modal. This multi-modality appears due to different discrete numbers of measurements. Figure 4.14 shows also an outlier in the evaluation of ENTPP-8. However, removing this value only changes the results in Table 4.2 marginally, by less than 0.5 s for each configuration.

In each configuration, the rollout path planner achieves on average the smallest time until localization. It can be seen that the stochastic extension of ENTPP is not necessarily better than the original path planner in every configuration. How-

**Table 4.3:** Mean number of required measurements for Scenario 2

| | Fast | Medium | Slow |
|---|---|---|---|
| Base policy | 2.410 | 2.410 | 2.410 |
| ENTPP | 2.758 | 2.758 | 2.758 |
| ENTPP-8 | 2.529 | 2.529 | 2.529 |
| Rollout | 2.530 | 2.308 | 2.229 |
| SOPT | 9.028 | 9.030 | 9.029 |

ever, since ENTPP does not completely capture the underlying problem, there is no reason why a better approximation of the expected entropy of the updated belief would lead to an improved performance. While in Scenario 1 ENTPP-8 performed better than ENTPP for most target distances, this does not seem to generalize for a scenario with a different target distribution. As SOPT requires on average a higher number of measurements in comparison to the other path planners (see Table 4.3), its relative performance strongly depends on the measurement duration. For the *fast* configuration it is comparable to the other methods, however, for the *slow* configuration it needs much more time than the other algorithms.

Figure 4.15 shows the selection of the first sensing action for the *fast* and *slow* configurations. As expected, there is no difference for the base policy, ENTPP, and ENTPP-8, since they do not consider the measurement time. Interestingly however, the other path planners also do not vary. SOPT uses the measurement duration mostly to decide on the number of measurements, then selects the measurement locations based on the required accuracy. If the measurements already are sufficiently accurate it only selects a single measurement for both measurement locations, but does not necessarily change the measurement location. Interestingly, there is also no difference in the first measurement location for the rollout path planner. However, in the course of the remaining measurements an adaptation takes place, as can be seen in Table 4.3. When the measurement duration is longer, the rollout path planner selects the measurement locations in way that overall less measurements are required. Also the measurement count for SOPT varies, however by a smaller amount.

**Table 4.4:** Mean planner computation time per step in seconds for Scenario 2

| | Fast | Medium | Slow |
|---|---|---|---|
| Base policy | 0.000 | 0.000 | 0.000 |
| SOPT | 0.001 | 0.000 | 0.000 |
| ENTPP | 0.545 | 0.553 | 0.548 |
| ENTPP-8 | 2.097 | 2.128 | 2.148 |
| Rollout | 2.032 | 2.117 | 2.117 |

Table 4.4 shows for each path planner the average computation time in a single decision step. It can be seen that the base policy and SOPT require a negligible amount. The ENTPP algorithm requires less computation time than the rollout algorithm even though it uses a larger $60 \times 60$ action grid than the $20 \times 20$ action grid of the rollout. ENTPP-8 requires approximately the same amount of planning time as the rollout algorithm, while still using a larger $40 \times 40$ action grid. Normalized by the size of the action grid, ENTPP requires 0.15 ms per action, ENTPP-8 1.328 ms

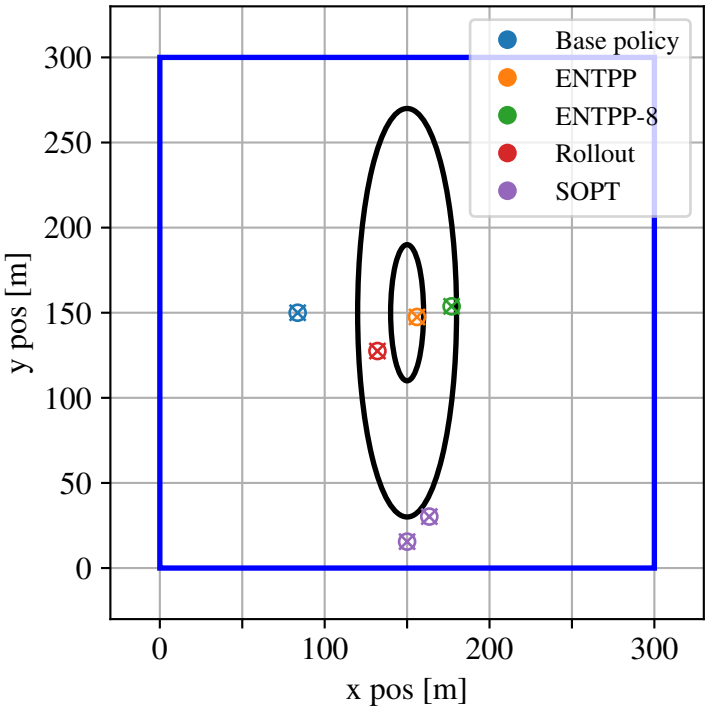



**Figure 4.15:** First sensing action per path planner for Scenario 2 with the fast configuration (circles) and the slow configuration (crosses). Note that in SOPT a single planner execution returns two measurement locations.

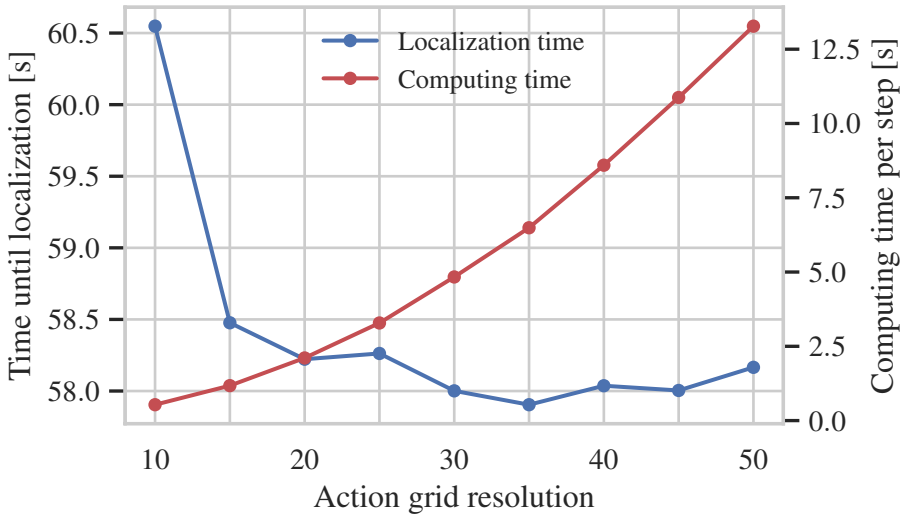



**Figure 4.16:** Effect of varying the action grid cell number $X^a = Y^a$ in Scenario 2, medium configuration. For each size of the action grid, 1000 MC runs are performed and the resulting time until localization is averaged. For a size of $X^a = Y^a = 20$ this matches the value in Table 4.2 with some small differences due to the MC sampling.

and the rollout algorithm 5.22 ms. The next paragraph analyses for the rollout algorithm the trade-off between performance and computation time for different sizes of the action grid. Different methods to speed up the evaluation of the rollout are analysed in Chapter 5 and in Chapter 6 a multi-threaded implementation is used to further reduce the computation time.

Figure 4.16 illustrates the results of varying the action grid resolution of the rollout path planner. While the extension of the $X^a \times Y^a$ action grid is kept constant, its resolution is increased by an increase of the cell number $X^a = Y^a$ in both dimensions. It can be observed that with increasing resolution of the action grid $\mathbb{A}_k$, the required time until localization decreases, until a $35 \times 35$ action grid. However, it increases again with further increase of the resolution. This effect can be explained by the hypothesis that the approximation of the action value with only eight deterministic samples captures the true action values sufficiently well, such that its minimum is a good sensing action and approaching the minimum of the action value approximation improves the result. However, it is still distinct from the true action value, which is why finding the real minimum of the approximation

does not lead to a truly optimal decision. In Chapter 5, we will further analyse how well different methods of computing the expected value relate to the approximative true minimum of $q^B$. Especially, we will see that by using either more deterministic samples as in these experiments, or other methods as *sequential halving*, it becomes more likely that the chosen action from the same $20 \times 20$ action grid is close to the true best action.

As expected, we see that the computation time increases with a higher action grid resolution. The computation time is measured as the CPU time on an *Intel Core i7-4770* with 3.40 GHz, using a Java-based implementation. It should be noted that the computing times are averages. Typically the first sensing actions take longer to compute than the later actions because the rollout is computed until localization. The rollout for the first action takes therefore longer to compute than if the target is localized in the next measurement step.

## 4.5 Conclusion

This chapter formulates the problem of localizing an RF emitting target using only bearing measurements with a mobile sensor system that needs to remain stationary for a period of time to take a measurement. To solve the problem, a path planner based on the policy rollout principle is created. This rollout path planner is tested in two different scenarios and shows promising results when compared with existing path planners from the literature.

Scenario 1 shows that no path planner is clearly better in every situation. It is possible to find for most path planners a target position where it shows the best results. For example, SOPT always takes the first measurement close to the initial platform position and, if the target is near this position, performs well. The entropy-based path planners always take their first measurement close to the centre of the uncertainty and, if the target is actually there, they perform well. While the rollout path planner is not the best for all positions, it is the best on several and performs well on all target distances.

The results of Scenario 1 indicate that it is useful to analyse the average performance of a path planner. This is done in Scenario 2. Here the belief corresponds to the true uncertainty in the scenario, therefore a path planner which uses the available information well, should lead to better results. In this comparison, the rollout path planner shows a better performance than each of the previously existing path planners for each configuration.

The results also show a consistent improvement of the rollout path planner to the base policy. While the base policy by itself does not consider the travel cost, these are included by the policy improvement step. This creates from the - in many cases worst performing - base policy the best path planner. This shows that by planning ahead, the path planner can achieve better results.

In total this chapter makes the following contributions:

1. A novel path planner for the stated problem, based on the policy rollout principle and deterministic samples
2. A comparison of this rollout path planner with existing path planners from the literature
3. An adaptive localizer, which uses a convex hull to focus the computation of a grid-based Bayes filter

Figure 4.16 shows that the parameters used in this chapter lead to significant computation times. A simple way to speed up the path planner would be parallelization. In the experiments in Chapter 6, a multi-threaded implementation of the rollout path planner will be used. But we want to analyse first whether the computation can be sped up by algorithmic improvements. We can identify a major source of inefficiency in the search for the optimal action: evaluating each action with the same amount of samples, even if the first samples might already indicate that it is inferior, seems like a waste. This begs the question: is there a better way to search for the optimal action value? We similarly might ask: are the deterministic samples really the best way to evaluate the expected value?

In the next chapter, we aim to answer these questions. We also analyse how the amount of computation time spent, measured in the number of rollouts, relates

to the performance for different ways to finding the minimum. This allows us to find a useful search method and parameters for an experimental evaluation. All this assumes that finding an action $\mathbf{a}$ with smaller cost in $\mathrm{Q}^B(b, \mathbf{a})$ also improves the actual performance of the resulting rollout path planner. Whether this is the case is also analysed in the next chapter.

# Chapter 5

# Efficient Online Policy Rollout

The previous chapter described a path planner based on the policy rollout algorithm. This path planner shows promising results, and improves on state of the art path planners from the literature. In this chapter, we further analyse two components of this planner: the search for the best action and the evaluation of the expected value. The contents of this chapter have previously been published in [Hoffmann et al., 2021].

Remember that the rollout path planner from the previous chapter selects the sensing action $\mathbf{a}_k \in \mathcal{A}_k \subseteq \mathbb{R}^2$ according to

$$\mathbf{a}_k = \underset{\mathbf{a} \in \mathbb{A}_k}{\operatorname{argmin}} \, \hat{q}^B(b_k, \mathbf{a}, \omega_{k1:R}) \, , \tag{5.1}$$

which is the action with the minimum sampled action value. Each action value is sampled by $R$ rollouts with the deterministic sample paths $\omega_{kj}$, $j \in \{1, ..., R\}$. This is an approximative solution to the more complex decision problem

$$\mathbf{a}_k^* = \underset{\mathbf{a} \in \mathcal{A}_k}{\operatorname{argmin}} \, q^B(b_k, \mathbf{a}) \tag{5.2}$$

of selecting the optimal action, which minimizes the true action value function $q^B$ over the continuous action space $\mathcal{A}_k$. This chapter explores alternatives to evaluate each action $\mathbf{a} \in \mathbb{A}_k$ with the same number of rollouts $R$ and sample paths derived from deterministic samples. Especially, we consider the following questions:

**The sampling of the action value.** What is the best way to sample the action value for a given action? This means for a given rollout $\hat{q}^B(b_k, \mathbf{a}, \omega)$, how should the sample path $\omega$ be chosen?

**The search for the optimal action.** How to choose the actions for which rollouts are performed? How many rollouts should be made for each action? How can we perform action search in a continuous action space?

Section 5.1 reviews existing solutions to these questions. Section 5.2 discusses different options to compute the sample paths. These are the already introduced deterministic samples, plain Monte Carlo (PMC), and common random numbers (CRNs), a technique for variance reduction. In Section 5.3 we introduce the idea of an *action selection algorithm*. This is an algorithm which proposes new actions $\mathbf{a}$ and sample paths $\omega$ for which a rollout should be performed. After performing a fixed number of rollouts, the action selection algorithm commits to the action $\mathbf{a}_k$. These algorithms are evaluated in Section 5.4. We are interested in two performance measures. The *optimization performance* measures how good an algorithm is in finding the minimum of the true action value function (5.2). The *localization performance* measures the effect of the action selection algorithm on the performance of the resulting rollout algorithm. This is done by the *time until localization* metric, defined in Chapter 4.

## 5.1 Comparison with existing work

In the literature on policy rollout, the search for the optimal action, i.e. the argmin in (5.1), is commonly performed by evaluation of each action from a finite set of actions. The expected value is computed for each action, for example with a fixed number of Monte Carlo samples. This not only requires a discretization if the action space is continuous, but also wastes computation time on performing rollouts for suboptimal actions.

Since it is a generic method, the policy rollout algorithm has been applied to a variety of problems. Early works consider the problem of playing Backgammon [Tesauro and Galperin, 1996], combinatorial optimization [Bertsekas et al., 1997]

and stochastic scheduling [Bertsekas and Castañón, 1999]. It has been used in several works in sensor management [Chong et al., 2008, 2009; Krakow et al., 2006; Zahedi et al., 2013; Ragi et al., 2015; Beyme and Leung, 2015; Jun and Jones, 2013; Charlish and Hoffmann, 2015], in particular for the activation of nodes in a sensor network [Saksena and Wang, 2008; He and Chong, 2004, 2006; Li et al., 2009] and sensor to target association [Zhang and Shan, 2015; Schneider and Chong, 2006]. Other problem settings encompass vehicle routing [Ulmer et al., 2019; Secomandi, 2003, 2001; Novoa and Storer, 2009; Goodson et al., 2013], inventory routing [Bertazzi et al., 2013], revenue management [Bertsimas and Popescu, 2003] and scheduling [McGovern and Moss, 1999].

The following two sections discuss those works in the literature, which differ from the simple approach that evaluates each action individually by a fixed amount of Monte Carlo samples.

### 5.1.1 Sampling of the action value

A main component of the rollout algorithm is the computation of the expected value of using action **a** and afterwards following the base policy, conditioned on the current belief.

In some problem domains an analytical solution can be computed [Secomandi, 2003, 2001; Bertazzi et al., 2013; Novoa and Storer, 2009; Goodson et al., 2013], typically via dynamic programming on the discrete state space. However, in sensor management, an exact evaluation is only possible in very specific cases [Jun and Jones, 2013], as the state space is usually continuous.

Often, the expected action value is computed via Monte Carlo sampling, which is almost exclusively the case in sensor management applications [Saksena and Wang, 2008; Ragi et al., 2015; Krakow et al., 2006; He and Chong, 2004, 2006; Li et al., 2009]. Monte Carlo evaluation is compared with exact analytical solutions in [Novoa and Storer, 2009] and [Goodson et al., 2013], which discusses the trade-off between solution quality and computation time. In [Novoa and Storer, 2009] no degradation of the solution is observed at all when using Monte Carlo evaluation instead of an exact solution.

When selecting between multiple actions, the objective is to select the action with the lower true action value. Monte Carlo methods to estimate an expected value always have a variance in their estimate and a higher variance leads to a higher probability of selecting the worse action. *Common random numbers* (CRNs) or *common random variables* are a technique for variance reduction [Rubinstein and Kroese, 2016, Ch. 5]. This technique is based on the fact that the exact value of $\mathbb{E}\left[\mathrm{Q}^B(b_k, \mathbf{a})\right]$ is not important, but only the question whether

$$\mathbb{E}\left[\mathrm{Q}^B(b_k, \mathbf{a}_1)\right] < \mathbb{E}\left[\mathrm{Q}^B(b_k, \mathbf{a}_2)\right] \tag{5.3}$$

for two actions $\mathbf{a}_1$ and $\mathbf{a}_2$. By using the same realizations of random variables to estimate both expected values, the sampling variances are correlated and the decision which is the truly smaller action value is more likely correct. In the rollout literature, this is also known as sampling of the Q-factor differences [Bertsekas, 1997]. Common random numbers have been used in several works on vehicle routing [Novoa and Storer, 2009; Goodson et al., 2013], however, have not been explicitly reported in the sensor management literature in the context of policy rollout. The works in [He and Chong, 2004, 2006; Krakow et al., 2006; Li et al., 2009] can be considered as using a form of CRNs, as they initialize the target state in the Monte Carlo rollouts using particles from a particle filter. Because each action is evaluated using the same particle set, the initial target state samples are the same for each action.

A different method is the computation of the expected action value with a set of representative fixed values for the random variables, also called scenarios [Bertsekas and Castañón, 1999]. One example for this approach is the work reported in [Zhang and Shan, 2015]. Here samples of the belief and the future measurements are deterministically created using a method similar to the unscented transform. The expected value is then computed using those samples. However, this only works for Gaussian probability densities. As a special case of deterministic samples, the expected values of the random variables can be used, for example in [Charlish and Hoffmann, 2015].

In Chapter 4, a generic method that suboptimally discretizes arbitrary probability densities [Klumpp and Hanebeck, 2008] was used to create a set of representative samples. In this chapter, we compare the effectiveness of independent samples, common random numbers and deterministic samples. While independent samples are often used, common random numbers are non-standard in sensor management, and the evaluation with deterministic samples is only used in a small number of works.

### 5.1.2 Search for the optimal action

Next to the computation of the expected value, another main component in evaluating (5.1) is the procedure to compute the argmin, which means to search for the action with minimum cost. In the vast majority of the literature, this is performed by evaluating every action in a finite action space. If the action values are determined via Monte Carlo sampling, an equal number of samples is commonly used for each action. For example [Saksena and Wang, 2008; McGovern and Moss, 1999; Ragi et al., 2015; Krakow et al., 2006; He and Chong, 2004, 2006; Li et al., 2009; Novoa and Storer, 2009; Goodson et al., 2013] all use Monte Carlo sampling with the same number of samples per action.

If the action space contains a large number of actions, evaluating each action might not be feasible. An option to speed up the search is to prune this space prior to the evaluation, which is done in [Zahedi et al., 2013; Schneider and Chong, 2006] and is also used in the previous chapter by limiting $\mathbb{A}_k$ to sensing actions close to the belief grid.

If it is possible to evaluate each action and sampling is used to estimate the action values, the search can be improved by non-uniformly allocating the total number of samples between the different possible actions. The idea is that it is more important to estimate the value of actions that are candidates of being the optimal action, instead of improving the estimate of clearly inferior actions. As this uses the results of previous action evaluations to improve the allocation of future samples, it can be described as *adaptive action evaluation*. Tesauro proposed to stop evaluation of an action once it becomes unlikely that it is the optimal one [Tesauro

and Galperin, 1996]. Optimization of the sample allocation has been performed by [Sun et al., 2008], using the optimal computing budget allocation (OCBA) [Chen and Lee, 2011] algorithm. The OCBA algorithm has a similar setting as the best arm selection problem in multi-armed bandits. It computes the sample mean and standard deviation for different actions using a fixed number of samples in a first phase and then uses those statistics to determine how often each action should be sampled in a second phase.

A method related to the online policy rollout method is classification-based policy iteration, which uses policy rollouts offline during training of a policy [Gabillon and Lazaric, 2010]. The algorithm performs rollouts for each state in a set of states, thereby deciding on an action for each state. This results in a dataset with tuples consisting of a state and an action from a finite action set. The policy improvement step of policy iteration is then performed by training a classificator on this dataset, thereby learning a mapping from state to action. This mapping is the policy in the next iteration. In [Gabillon and Lazaric, 2010], a multiple multi-armed bandit method is used to determine how often a rollout is performed for each state and each action.

Another method, related to the policy rollout method is Monte Carlo tree search (MCTS) [Browne et al., 2012]. MCTS iteratively builds a search tree, focusing on those nodes which are expected to have the highest values. The estimation of the value of a node is performed by rollouts of either an existing or a random policy. The decision which node should be expanded next is modelled as a multi-armed bandit problem, using an adaption of the upper confidence bounds (UCB) algorithm for trees (UCT) [Kocsis and Szepesvari, 2006]. MCTS is mostly applied to discrete state spaces and actions, however, bandit algorithms for continuous action spaces [Bubeck et al., 2011] have been used to perform continuous action selection with a discrete state space [Mansley et al., 2011]. In MCTS the multi-armed bandit formulation is based on the classical formulation, which describes a trade-off between exploration and exploitation. This is because during creation of the search tree, it is both important to expand the subtree of good child nodes as well as to explore

the subtrees of not yet well explored but worse child nodes. In comparison, for a use with the policy rollout algorithm, the best arm selection problem discussed in Section 2.1 is more useful.

### 5.1.3 Contributions of this chapter

While some publications using policy rollout describe optimized action selection algorithms, they are in the minority. A comparison of different action selection algorithms and sampling methods also was not yet performed in the literature and none of the policy rollout works considered a continuous action space. This has only been found in related algorithms like MCTS.

This chapter follows the idea of considering the policy rollout action selection step as a function minimization problem, discussed in Section 2.1. This allows us to use established methods to solve such a problem in the policy rollout context, for the continuous and discrete case. While the algorithms are not novel by themselves, with the exception of uniform allocation, their use as action selection algorithm in the policy rollout is novel. Different methods are compared and their performance on the problem is analysed.

## 5.2 Sampling methods

The result of a rollout $j$ for action $\mathbf{a}$ at decision step $k$ is determined by the random variables

$$\Omega_{k\mathbf{a}j} = (\mathrm{X}^t_{k\mathbf{a}j}, \mathrm{W}^m_{k\mathbf{a}j[k+1]}, \mathrm{W}^m_{k\mathbf{a}j[k+2]}, ..., \mathrm{W}^m_{k\mathbf{a}j\mathrm{K}}) \tag{5.4}$$

whose realization

$$\omega_{k\mathbf{a}j} = \left(\mathbf{x}^t_{k\mathbf{a}j}, w^m_{k\mathbf{a}j[k+1]}, w^m_{k\mathbf{a}j[k+2]}, ..., w^m_{k\mathbf{a}j\mathrm{K}}\right) . \tag{5.5}$$

is called a *sample path*. For the creation of these sample paths, we compare three different approaches, plain Monte Carlo, common random numbers, and the deterministic sampling approach.

In the actual implementation, the sample path $\omega_{k\mathbf{a}j}$ is represented as a tuple $(\omega^x_{k\mathbf{a}j}, \omega^y_{k\mathbf{a}j}, \omega^m_{k\mathbf{a}j})$, where $(\omega^x_{k\mathbf{a}j}, \omega^y_{k\mathbf{a}j})$ is the sampled target position and

$\omega^m_{k\mathbf{a}j} \in \mathbb{Z} \cup \{\text{None}\}$ a seed for the measurement noise pseudorandom number generator or an indication that the measurement noise should be zero.

### 5.2.1 Plain Monte Carlo

In the plain Monte Carlo (PMC) sampling method each sample path is independent. This means that $\Omega_{k\mathbf{a}j}$ and $\Omega_{k\mathbf{a}'j'}$ are independent when either $\mathbf{a} \neq \mathbf{a}'$ or $j \neq j'$ for actions $\mathbf{a}, \mathbf{a}'$ and rollouts $j, j'$. In the implementation this means that the sampled target position $\omega^x_{k\mathbf{a}j}, \omega^y_{k\mathbf{a}j}$ and seed $\omega^m_{k\mathbf{a}j}$ is different for each rollout.

### 5.2.2 Common random numbers

Common random numbers (CRNs) [Rubinstein and Kroese, 2016, Ch. 5] can be used to compare alternatives. With this method, the rollouts for different actions are performed using the same sample paths. This induces a correlation between the rollouts, which reduces the variance of the relative error in the action value estimates. Note that the absolute error is not reduced, however, only the relative error is important to make the right decision.

In this thesis CRNs are implemented, such that if $R$ rollouts are executed for two different actions $\mathbf{a}$ and $\mathbf{a}'$, we have

$$\Omega_{k\mathbf{a}j} = \Omega_{k\mathbf{a}'j} \tag{5.6}$$

for the same $j$-th rollout, $j \in \{1, \ldots, R\}$. This means that the target position

$$\mathrm{X}^t_{k\mathbf{a}j} = \mathrm{X}^t_{k\mathbf{a}'j} \tag{5.7}$$

and measurement noise

$$\mathrm{W}^m_{k\mathbf{a}ji} = \mathrm{W}^m_{k\mathbf{a}'ji} \tag{5.8}$$

are the same for the $j$-th rollout, different actions $\mathbf{a}$ and $\mathbf{a}'$ and future measurement step $i$. In the implementation this means that the sampled target position $\omega^x, \omega^y$ and seed $\omega^m$ are the same for those rollouts. Note that $\mathrm{X}^t_{k\mathbf{a}j}$ and $\mathrm{W}^m_{k\mathbf{a}ji}$ are still randomly sampled, but only once for each rollout number $j$ and measurement step $i$. We refer to both CRNs and PMC as Monte Carlo approaches.

### 5.2.3 Deterministic samples

We use the same deterministic sampling algorithm from [Klumpp and Hanebeck, 2008], previously described in Section 4.3 to sample from the target state. The algorithm choses the samples, such that they provide a good representation of the belief for the given number of $R$ samples. This implies that the number of samples $R$ must be known when the samples are computed and that the realizations $\omega_{k[1:R]}$ of the random variables $\Omega_{k[1:R]}$ are computed in a single execution of the algorithm. Especially, this means that they are not computed iteratively and computing $R-1$ sample paths $\omega_{k[1:R-1]}$ and one additional sample path $\omega_{kR}$ is a worse representation of the uncertainty than computing directly $R$ sample paths. Section 4.3 describes that the weights of the deterministic samples are only uniform if $R$ is a power of two. Therefore, the evaluation below only uses powers of two.

As in Chapter 4, the measurements are approximated without zero measurement noise

$$\mathrm{W}^{m}_{k\mathbf{a}ji} = 0 \,. \tag{5.9}$$

Because the sample path realizations are deterministic, they are the same when the same number of rollouts $R$ for different actions are performed in the same belief. Similar to CRN this introduces a correlation between the rollouts for different actions.

In the implementation of the policy rollout with deterministic samples, the target position $\omega^x, \omega^y$ is determined by the algorithm [Klumpp and Hanebeck, 2008] and $\omega^m =$ 'None' indicates that no measurement noise should be sampled.

## 5.3 Action selection algorithms

To determine the optimal action $\mathbf{a}^*$, the action that minimizes the true action value function $q^B$ for a given belief $b$ needs to be found. However, it is not feasible to directly evaluate $q^B$ but only indirectly by sampling individual rollouts $\hat{q}^B$. This chapter formulates adaptive action evaluation in a policy rollout algorithm as the interaction between an *action selection algorithm* and the actual policy rollout component.

Based on the current belief, the action selection algorithm determines the action $\mathbf{a}$ that should be evaluated and the sample path $\omega$ that should be used. The policy rollout component computes the corresponding action value $\hat{q}^B(b_k, \mathbf{a}, \omega)$, using Algorithm 1. The rollout result $\hat{q}^B(b_k, \mathbf{a}, \omega)$ is a deterministic function of the parameters and its value is returned to the action selection algorithm. Each call of this function is equivalent to a single rollout. The action selection algorithm then uses this rollout result and all previous rollout results to compute a new action $\mathbf{a}'$ and sample path $\omega'$ which should be evaluated next. This is repeated until the action selection algorithm decides on an action $\mathbf{a}_k$ which should be executed by the rollout path planner. The procedure is visualized in Figure 5.1. The concept of an action selection algorithm serves as a common interface to apply the stochastic function minimization algorithms discussed in Section 2.1 to a policy rollout algorithm.

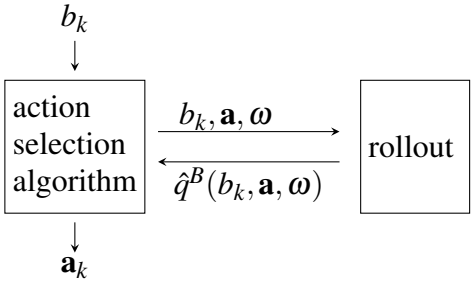



**Figure 5.1:** Visualization of an action selection algorithm.

The sample path can be created by the different sampling methods described in the previous sections: plain Monte Carlo samples, common random numbers, or deterministic samples. If an action selection algorithm supports multiple sampling methods, we refer to them as *variants* of this algorithm.

To compare the action selection algorithms, it is useful to consider their performance based on the total amount of rollouts they require. We define the *computational budget N* as an upper limit on the calls of the function $\hat{q}^B$. The number of times this function is called for a given action selection algorithm is called *total number of rollouts*. Therefore, for a given computational budget we want to select the best performing action selection algorithm whose total number of rollouts are

less than the computational budget. In this thesis we assume a constant computational budget for each decision step $k$.

The combination of sampling method, computational budget, and additional algorithm-specific parameters is called the *configuration* of an action selection algorithm.

### 5.3.1 Uniform allocation

The simplest action selection algorithm is to perform a fixed number of rollouts for each action in a finite action space. We use the same discretized action space $\mathbb{A}_k$ as in Section 4.3 and set the number of rollouts per action to

$$R_k = \left\lfloor \frac{N}{|\mathbb{A}_k|} \right\rfloor \tag{5.10}$$

for each action $\mathbf{a}$. Then the action

$$\mathrm{A}_k = \operatorname*{argmin}_{\mathbf{a} \in \mathbb{A}_k} \hat{q}^B(b_k, \mathbf{a}, \Omega_{k\mathbf{a}[1:R_k]}) \tag{5.11}$$

is selected by performing $R_k$ rollouts for each action. Because of the sampling, the result is a random variable $\mathrm{A}_k$ and not a deterministic action $\mathbf{a}_k$. The realizations of $\Omega_{k\mathbf{a}j}$ are determined either by plain Monte Carlo, common random numbers or deterministic samples. Note, that this is also the algorithm used in the previous chapter.

### 5.3.2 Multi-armed bandits

Section 2.1 discussed the best arm selection problem in multi-armed bandits. This problem consists of selecting the action with the lowest true cost from a finite set of alternatives, where only samples of the true cost of each alternative are known.

There is a clear connection between the best arm selection problem and the problem of action selection in a sampling-based policy rollout algorithm. In both cases, there are multiple candidate actions, and the value of each action is only available by repeatedly performing a rollout or repeatedly pulling an arm. Each additional rollout improves the estimate of the true action value. Different to the

classical bandit formulation, in the best arm selection problem and in the policy rollout case it is not important how much cost or reward is gained during the evaluation of the actions. Only the cost of the selected action is important.

Contrary to uniform allocation, such an algorithm evaluates different actions with a different number of rollouts. Similar to uniform allocation, the algorithm requires a finite action space, for which we use the same discretization $\mathbb{A}_k$.

The best arm selection algorithm used in this thesis is called sequential halving and was already described in Section 2.1. We use an implementation with plain Monte Carlo sampling, as well as one with common random numbers in the rollouts.

In each iteration $l$ of the sequential halving algorithm, each remaining action is evaluated with the same number of rollouts $R_{kl}$. For the variant with CRN, therefore, $R_{kl}$ random variables $\Omega_{k\mathbf{a}jl}$, $j \in \{1,\ldots,R_{kl}\}$ need to be sampled, with

$$\Omega_{k\mathbf{a}jl} = \Omega_{k\mathbf{a}'jl} \tag{5.12}$$

for actions $\mathbf{a}, \mathbf{a}'$.

Using the deterministic sampling method with sequential halving is not reasonable, because it is not possible to increase the number of samples adaptively. Assume that

$$\mathbb{S}_R = \{(\omega_1^x, \omega_1^y), ..., (\omega_R^x, \omega_R^y)\} \tag{5.13}$$

is a set containing $R$ samples representing possible target positions of a belief. Increasing the sample size to $\hat{R}$ with $\hat{R} > R$ leads to a different set of target position samples

$$\mathbb{S}_{\hat{R}} = \{(\hat{\omega}_1^x, \hat{\omega}_1^y), ..., (\hat{\omega}_{\hat{R}}^x, \hat{\omega}_{\hat{R}}^y)\} \,. \tag{5.14}$$

However, it is always the case that $\mathbb{S}_R \not\subseteq \mathbb{S}_{\hat{R}}$, because the algorithm works by splitting the existing samples $(\omega_j^x, \omega_j^y)$. Figure 4.6 in the previous chapter shows the same density approximated with 1,2,4, and 32 samples. It can be seen that the samples of one approximation are not contained in the approximation with a higher number of samples. In addition, while $\mathbb{S}_R$ is a good approximation of the density with $R$

samples and $\mathbb{S}_{\hat{R}}$ a good approximation with $\hat{R}$ samples, $\mathbb{S}_R \cup \mathbb{S}_{\hat{R}}$ is not necessarily a good approximation for $R + \hat{R}$ samples.

The sequential halving algorithm has the requirement that it is possible to decide on the number of samples adaptively. Let $R$ be the number of rollouts per action in the first iteration and $\hat{R} - R$ in the second iteration. Due to the argument above, it is not possible to simply create $\hat{R} - R$ additional samples, as the distribution would be different and suboptimal to directly having created $\hat{R}$ samples. The advantage of the deterministic sampling approach, to cover the density as efficiently as possible with the given number of samples, would be lost if not all samples are created in the same step. In comparison, each additional Monte Carlo sample is independent from the previous ones and it is simple to increase the number of samples adaptively.

### 5.3.3 Quadrant search

Quadrant search is the method of restricting the search iteratively to the most promising quadrant and has been used for sensor path planning before [Hernandez, 2004]. In this section the algorithm is applied to the policy rollout case.

At the start of the algorithm, the action values on a $3 \times 3$ grid are sampled by performing $R$ rollouts each, effectively dividing the action space into four quadrants. Then for each quadrant, the mean of the sampled action values at its four corners is computed. Based on these values, the search focuses on the quadrant with the lowest mean action value. In this quadrant, the action values of additional five actions are sampled, in the centre and the middle of each border. Therefore, this quadrant now contains a smaller $3 \times 3$ grid. Then a quadrant from this smaller $3 \times 3$ grid is chosen. This subdivision is repeated for a fixed number of $L$ iterations, leading to a used computational budget of

$$N = R \cdot (9 + L \cdot 5) \tag{5.15}$$

rollouts.

Figure 5.2 visualizes this process. In the first iteration, rollouts are made for the actions $A - I$. Then the quadrant averages are compared and it is determined that the average action value

$$\frac{1}{4} \sum_{\mathbf{a} \in \{B,C,E,F\}} \hat{q}^B(b_k, \mathbf{a}, \Omega_{k\mathbf{a}[1:R]}) \tag{5.16}$$

of the upper right quadrant is less than the corresponding averages for the other quadrants. Therefore, in the second iteration the action values at $J, K, L, M, N$ are sampled. The next iteration would select a quadrant of $(B, C, E, F)$, for example $(L, M, N, F)$, and repeat the subdivision.

We implemented this method in a variant with common random numbers, as well as a variant based on deterministic samples. In both cases $\Omega_{k\mathbf{a}[1:R]}$ is not only the same for each action $\mathbf{a}$, but also in each iteration of quadrant search. With the example of Figure 5.2 this means that all actions $\mathbf{a} \in \{A, ..., N\}$ are evaluated with the same sample paths. This is required to use CRN or deterministic samples, because the action values of different iterations are compared with each other. For example, the sub-quadrant $BJKL$ is compared with $KLEN$, where $B$ and $E$ are from the first iteration, and the remaining points from the second iteration. In comparison, an implementation with plain Monte Carlo samples would use a different sample path for each action.

In the scenarios evaluated in this chapter, the scenario area is rectangular. Therefore, the action space $\mathcal{A}_k$ constructed by (4.28) is rectangular as well. The initial area on which the $3 \times 3$ grid is placed is based on the extension of $\mathcal{A}_k$ and therefore has the same limits as the action grid of the uniform allocation and sequential halving approaches.

### 5.3.4 Gradient-based algorithms

Gradient-based algorithms are commonly used in function minimization problems, as discussed in Section 2.1. We apply two gradient-based methods to the problem: stochastic gradient descent and the BFGS algorithm. Similar to the base policy in

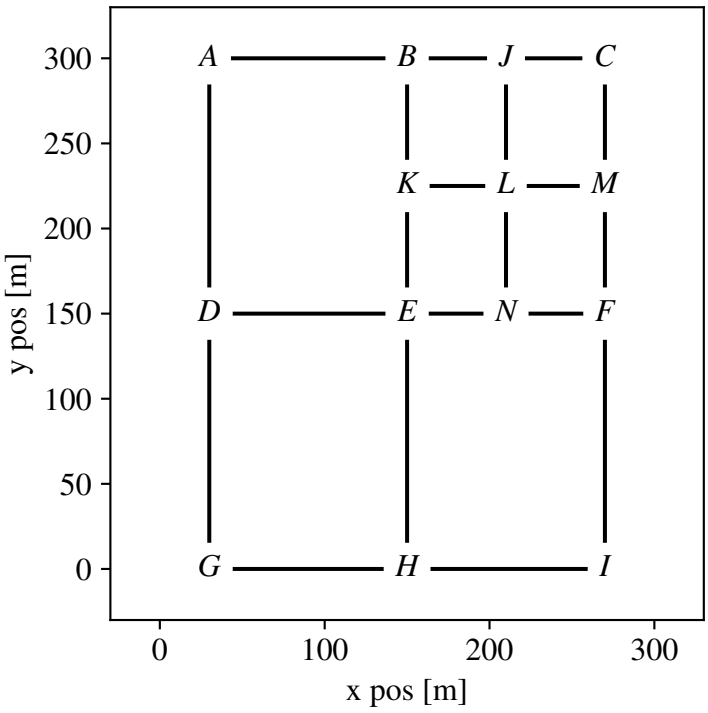



**Figure 5.2:** Visualization of the quadrant search method.

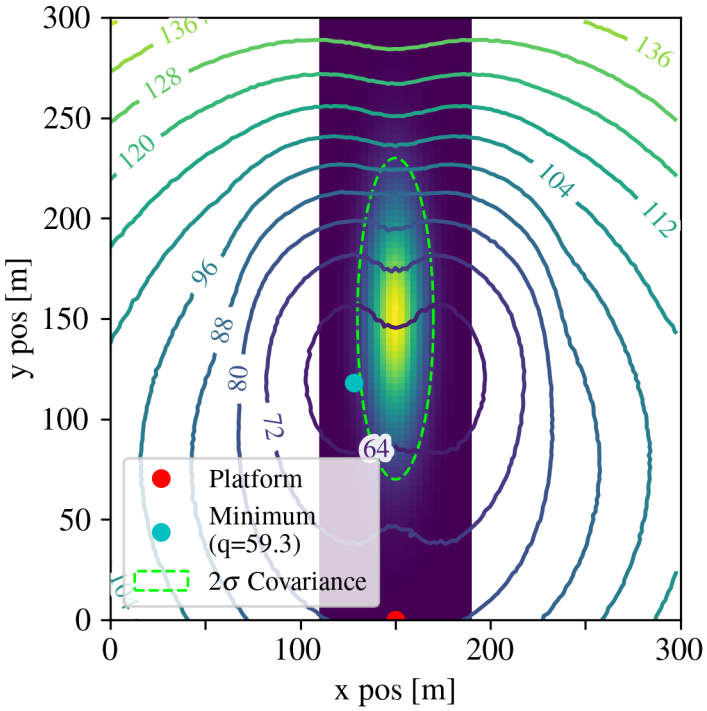
Section 4.4, we use an unconstrained sensor state space with $\mathcal{X}^s = \mathbb{R}^2$ for those two algorithms, because they solve an unconstrained optimization problem.

Gradient-based algorithms work well if the function does not have too many local minima and saddle points. Figure 5.3 shows a close approximation of the true action value function $q^B$ for two different beliefs, computed using a sufficient number of samples (see Section 5.4.1). The function appears sufficiently smooth under a visual inspection, and typically has two local minima.

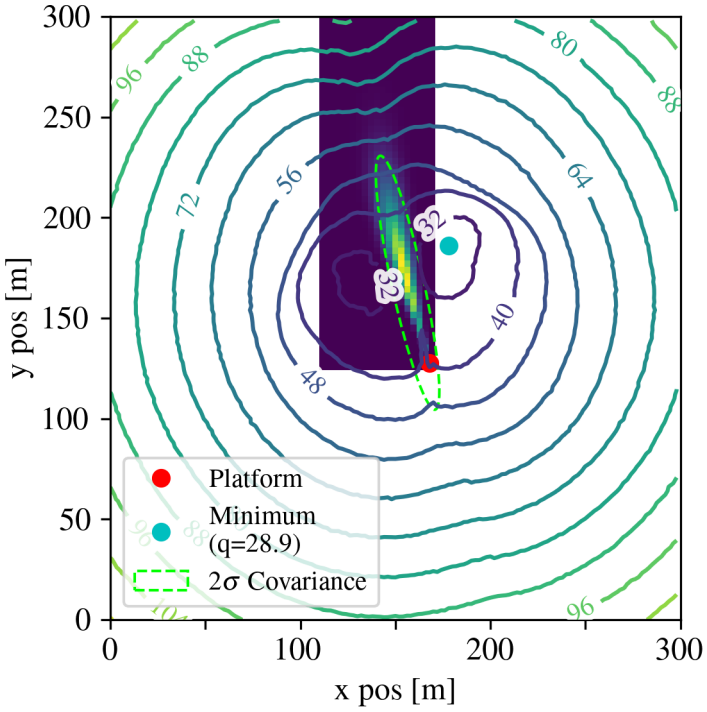
Stochastic gradient descent improves an initial action selection $\mathbf{a}_{k1}$ using samples $\mathrm{G}_{kl}$ of the gradient in the following iteration

$$\mathrm{A}_{k[l+1]} = \mathrm{A}_{kl} - \eta_l \mathrm{G}_{kl}(b_k, \mathrm{A}_{kl}) \tag{5.17}$$

where $\eta_l$ is the step size and $l$ the iteration. The initial action $\mathrm{A}_{k1} = \mathbf{a}_{k1}$ is determined by the base policy $\pi^B$. As the base policy is a heuristic for the problem, this is a reasonable initialization for the iteration.




(a) Belief 0




(b) Belief 16

**Figure 5.3:** Estimation of the true action value $q^{\pi}$ using a sufficiently high number of roll-outs. The heat map shows the belief, while the contours represent the action values. The $2\sigma$ confidence ellipsoid for the Gaussian approximation $(\tilde{\mathbf{x}}^t, \tilde{\mathbf{P}}^t)$ of the belief is shown in green.

There is no straightforward way to compute the gradient of the policy rollout analytically, therefore it is approximated by finite differences. The gradient $\mathrm{G}_{kl}$ at position $\mathrm{A}_{kl}$ is computed by

$$\mathrm{G}_{kl} = \frac{1}{2a^{\mathrm{FD}}} \cdot \begin{pmatrix} \hat{q}^B(b_k, \mathrm{A}_{kl} + a^{\mathrm{FD}} e_0, \Omega_{kl[1:R]}) - \hat{q}^B(b_k, \mathrm{A}_{kl} - a^{\mathrm{FD}} e_0, \Omega_{kl[1:R]}) \\ \hat{q}^B(b_k, \mathrm{A}_{kl} + a^{\mathrm{FD}} e_1, \Omega_{kl[1:R]}) - \hat{q}^B(b_k, \mathrm{A}_{kl} - a^{\mathrm{FD}} e_1, \Omega_{kl[1:R]}) \end{pmatrix}, \tag{5.18}$$

where $e_0 = (1,0)^{\mathrm{T}}$ and $e_1 = (0,1)^{\mathrm{T}}$ are the unit vectors for the $x$ and $y$ dimension, and $a^{\mathrm{FD}}$ is the size for the finite differences in each direction. The action values for an iteration $l$ are sampled with common random numbers, but different iterations of the algorithm use different sample paths. The finite difference step $a^{\mathrm{FD}} = 20$ is rather large because, although $q^B$ appears to be rather smooth, this does not necessarily need to be the case for $\hat{q}^B$, averaged over only a small number of rollouts. To make sure discontinuities in $\hat{q}^B$ do not lead to an estimate of the gradient too different than the true value, it is averaged over a larger area.

The step size schedule consists of an exponential decay, with

$$\eta_l = 20 \cdot \exp\left(-4 \frac{l-1}{L-1}\right). \tag{5.19}$$

With $L$ iterations $l \in \{1, \dots L\}$, the above step size schedule interpolates between $\eta_1 = 20$ and $\eta_L = 20 \cdot \exp(-4) \approx 0.366$.

When deterministic samples $\omega_{k[1:R]}$ are used, the function

$$\mathbf{a} \mapsto \hat{q}^B(b_k, \mathbf{a}, \omega_{k[1:R]}) \tag{5.20}$$

is a deterministic function, motivating the use of standard optimization methods. We use BFGS, which is an effective quasi-Newton method and was previously described in Section 2.1. For BFGS the same initialization, finite difference method and finite difference size was used, and the gradient was computed as in (5.18). The only difference is that the sample paths $\omega_{k[1:R]}$ are based on deterministic samples and are the same for each BFGS iteration. The algorithm is terminated once a predefined limit $\lfloor N/R \rfloor$ of calls to the function (5.20) is reached, including those to

estimate the gradient. The total number of rollouts then consists of this limit multiplied by the rollouts per action $R$. We use the implementation from JSAT [Raff, 2017], version 0.0.9, for the BFGS algorithm.

## 5.4 Evaluation

In this section two metrics are used for evaluating the different action selection algorithms. The *optimization performance* considers how much worse the true action value of the selected action is compared to the true action value of the optimal action. For this, an approximation of the true action value function is computed, using a high number of samples. The *localization performance* instead quantifies the effectiveness of the resulting rollout path planner for emitter localization, using the time until localization metric.

### 5.4.1 The true action value function

We evaluate the different action selection algorithms based on how well they can find the minimum of the true action value function $q^B$. As this function is not known, a close approximation of it was computed for a fixed set of beliefs $\mathbb{B}$. Then the minimization methods were tested on this set of beliefs.

To create the belief set $\mathbb{B}$, Scenario 2 (see Figure 4.8b) was simulated 20 times with the rollout path planner from Chapter 4 and the parameters from Table 4.1. Each Monte Carlo run $m, 1 \leq m \leq 20$, led to a sequence of beliefs $b_{m0}, b_{m1}, ..., b_{m\mathrm{K}_m}$, where $b_{m0}$ is the initial belief and the target is localized in belief $b_{m\mathrm{K}_m}$. The initial belief $b_{m0}$ is the same for every $m$ and therefore only included once. The final beliefs $b_{m\mathrm{K}_m}$ are also omitted, because no further action is necessary here. This leads to the belief set

$$\mathbb{B} = \{b_{10}\} \cup \{b_{mk} \mid 1 \leq k < \mathrm{K}_m, 1 \leq m \leq 20\} \,. \tag{5.21}$$

In total the belief set $\mathbb{B}$ contains 25 distinct beliefs.

For an approximation of the true action value function $q^B$, the action space was discretized on the whole scenario area with a $150 \times 150$ action grid, i.e. with 2 m distance between the actions. For each action $\mathbf{a}$ and belief $b \in \mathbb{B}$ the action

**Table 5.1:** Action selection algorithms with different sampling methods

| | PMC | CRNs | Det. |
|---|---|---|---|
| Uniform allocation | $\oplus$ | $\oplus$ | $\oplus$ |
| Sequential halving | $\oplus$ | $\oplus$ | |
| Quadrant search | $+$ | $\oplus$ | $\oplus$ |
| Stochastic gradient descent | $+$ | $\oplus$ | |
| BFGS | | $+$ | $\oplus$ |
| Possible variant | $+$ | | |
| Evaluated variant | $\oplus$ | | |

value was sampled using sufficiently many plain Monte Carlo samples such that the standard error of the mean

$$\mathrm{SEM}(b,\mathbf{a}) = \frac{\mathrm{Std}(\{\ \hat{q}^B(b,\mathbf{a},\Omega_{b\mathbf{a}j})\ \}_{1\leq j\leq \mathrm{R}_{b\mathbf{a}}})}{\sqrt{\mathrm{R}_{b\mathbf{a}}}} \tag{5.22}$$

was below 0.1. Here Std computes the standard deviation of all sampled action values $\hat{q}^B(b,\mathbf{a},\Omega_{b\mathbf{a}j})$ for action $\mathbf{a}$. The total number of rollouts is denoted by $\mathrm{R}_{b\mathbf{a}}$ and was around $2000-20\,000$ samples, depending on the action and belief. The resulting action values can be seen in Figure 5.3 for two exemplary beliefs. While it is technically only an approximation, it is sufficiently accurate that we refer to the results as the true action value function $q^B$.

With $\mathbf{a}_b^*$ we denote the action on the $150\times150$ action grid that minimizes $q^B$ for belief $b$. Again this is only an approximation, but under a visual inspection of the function $q^B(b,\mathbf{a})$ on the $150\times150$ action grid, it is not expected that the true optimum is significantly different.

It can be seen that the function $q^B$ has typically two minima, which are approximately symmetric to the major axis of the Gaussian approximation $(\tilde{\mathbf{x}}^t,\tilde{\mathbf{P}}^t)$ of the belief. This is similar to the points $M_1$ and $M_2$ in the base policy (see Figure 4.4a). However, the optimal action $\mathbf{a}_k^*$ is sometimes closer to the point estimate $\tilde{\mathbf{x}}^t$ of the target than the action chosen by the base policy. In addition, the optimal action is not always placed on a direct extension of the minor axis.

The rollout path planner was executed on each belief with the action selection algorithm variants shown in Table 5.1. Variants with deterministic samples were executed once, those with plain Monte Carlo or CRN 100 times. The index of the Monte Carlo run is $m$ and the chosen action in this run on belief $b$ is $\mathbf{a}_{bm}$. The action values $q^B$ of each belief $b$ were evaluated on a $150 \times 150$ grid, but the chosen action $\mathbf{a}_{bm}$ is not necessarily on one of the grid points. Instead, the action is typically inside a grid cell. The approximately true action value $q^B(b, \mathbf{a}_{bm})$ is computed by bilinear interpolation of the action values at the corners of the cell.

For each action selection algorithm of the rollout path planner, the *mean distance to optimum* is defined as

$$d^o = \frac{1}{|\mathbb{B}|M} \sum_{b \in \mathbb{B}} \sum_{m=1}^{M} q^B(b, \mathbf{a}_{bm}) - q^B(b, \mathbf{a}_b^*) \tag{5.23}$$

which is the difference between the value of the chosen action and the one of the optimal action, averaged over multiple Monte Carlo runs and different beliefs. Here $M$ is the number of Monte Carlo runs for the method, which was either 1 or 100. Note that $\mathbf{a}_b^*$ does not necessarily need to be unique, but the minimal action value $q^B(b, \mathbf{a}_b^*)$ is unique. The mean distance to optimum is a metric for the *optimization performance* of an action selection algorithm, which quantifies how well it can find the minimum of $q^B$.

### 5.4.2 Optimization performance

For improved clarity, the presentation of the results is separated into three sections, where the first discusses uniform allocation and sequential halving, the second quadrant search, and the third the gradient-based methods. The action selection algorithms are split up in this way as the first two consider each action independently and therefore, are also feasible for arbitrary action spaces. The second and third set of algorithms assume that the action value function is continuous, which means that performing a rollout for an action also gives information about nearby actions.

The results are presented as the mean distance to optimum $d^o$, defined in (5.23), achieved for a total number of rollouts. Each data point for an action selection

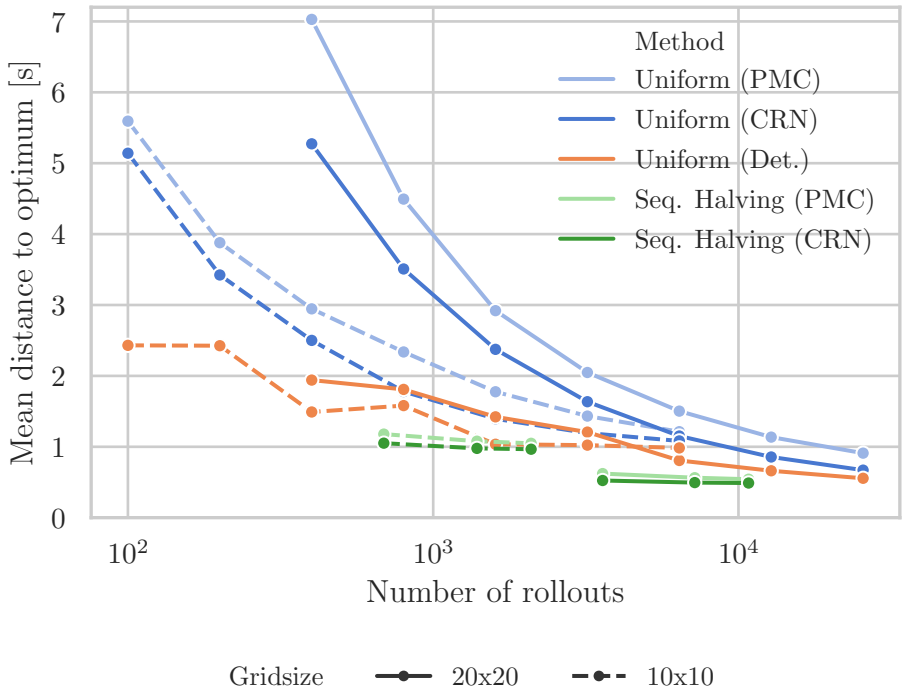



**Figure 5.4:** Optimization performance of uniform allocation and sequential halving.

algorithm corresponds to a single configuration, consisting of the sampling method, computational budget, and other algorithm-specific parameters. A configuration is considered *Pareto optimal*, if no other configuration achieves a better optimization performance with a less or equal number of rollouts.

## Uniform allocation and sequential halving

Figure 5.4 shows the results of uniform allocation and sequential halving. Note that in this and all similar plots the x-axis is logarithmic. Uniform allocation uses a $10\times10$ and $20\times20$ action grid, and the number of rollouts per action is varied over powers of two. This means the first data point of each line corresponds to a single rollout per action and a total number of either 100 $(=10\cdot10\cdot1)$ or 400 $(=20\cdot20\cdot1)$ rollouts. The following data points correspond to $2,4,8,\ldots$ rollouts per action.

Sequential halving is evaluated on the same action grids with computational budgets $N$ chosen as multiples of $100\cdot\lceil\log_2(100)\rceil=700$ for the $10\times10$ grid and as multiples of $400\cdot\lceil\log_2(400)\rceil=3600$ for the $20\times20$ grid. The exact numbers are

given in Table 5.2. As indicated in the table and described in Section 2.1, sequential halving uses a slightly lower number of rollouts than the computational budget.

**Table 5.2:** Computational budget for sequential halving.

| Grid | Computational budget | Used number of rollouts |
|---|---|---|
| 10×10 | 700 | 689 |
| | 1400 | 1391 |
| | 2100 | 2093 |
| 20×20 | 3600 | 3589 |
| | 7200 | 7191 |
| | 10800 | 10793 |

For uniform allocation, it can be seen that using common random numbers is strictly better than plain Monte Carlo samples. It can also be seen that the variant with deterministic samples achieves better results than using common random numbers. However, while the Monte Carlo sampling methods improve monotonically for increasing number of rollouts, the same is not true for the deterministic samples. Although with a higher number of rollouts the performance tends to improve, sometimes deterministic samples lead to a worse result for a higher number of rollouts. This is likely because the deterministic samples are a suboptimal approximation method for the belief about the target position. It should be noted that because the samples are deterministic, the chosen action is also deterministic and therefore this variant does not have the averaging of multiple Monte Carlo runs that is present with PMC and CRNs.

It can also be seen that for a small computational budget $N$, it is better to have a smaller action space with more rollouts per action than a larger action space with potentially better actions but where the number of rollouts per action is smaller. However, for a higher computational budget the larger action space shows better results.

For the considered parameters, sequential halving shows consistently better results than uniform allocation because it can focus on the most promising actions. Each evaluated configuration with CRNs is Pareto optimal. Uniform allocation

with 64 deterministic samples on the $20 \times 20$ grid shows approximately the same optimization performance as sequential halving with CRNs and a budget of 10 800. However, it requires more than twice the number of rollouts (25 600 vs 10 793). A disadvantage of sequential halving is that it requires a higher minimal computational budget (see Section 2.1.3) than uniform allocation.

These action selection algorithms are fundamentally limited, in that they will not be able to select an action not in their discretized action space. The optimal action $\mathbf{a}_b^*$ is computed over an action grid with a very high resolution (see Section 5.4.1) and does not necessarily have to be on the $10 \times 10$ or $20 \times 20$ action grid. Therefore, these algorithms likely would not reach a zero distance to optimum in Figure 5.4, even with a high amount of rollouts.

Finally, the base policy itself would score 2.98 s on this plot, which is better than some configurations. This shows that the rollout improvement property cannot be achieved if the action selection algorithm does not perform well.

## Quadrant search

Figure 5.5 shows the results of quadrant search. For a given number of iterations, we vary the number of rollouts per action, again being powers of two. For example, the first data point with a single rollout per action and three iterations corresponds to a total of $9 + 3 \cdot 5 = 24$ rollouts.

For an arbitrary action value function, this action selection algorithm is at the risk of focusing on the wrong quadrant because in general the optimal action is not guaranteed to be in the quadrant whose corners have on average the best action values. Since the algorithm makes a hard decision on the quadrant, it would not be able to recover from such a choice. Therefore, even with knowledge about the true action values at the corners, the algorithm would not necessarily select the optimal action. However, this does not seem to happen in this application and the action selection algorithm achieves a good performance for a small computational budget.

Similar to the case of uniform allocation, the variant with deterministic samples is better than the one with common random numbers.

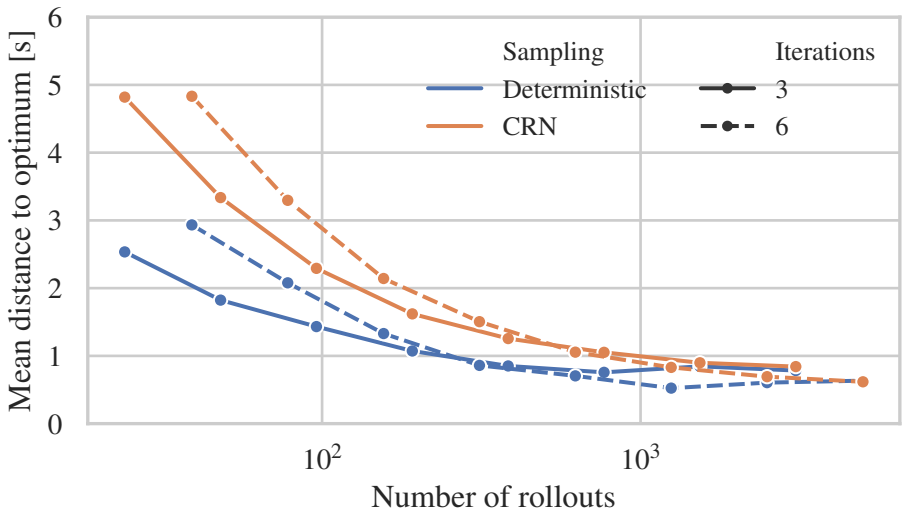



**Figure 5.5:** Optimization performance of quadrant search.

## Gradient-based algorithms

Figure 5.6 shows the results of stochastic gradient descent (SGD) and BFGS. For SGD the number of iterations $L \in \{5, 10, 25, 50, 100, 150, 200\}$ is varied, with a fixed number of $R$ rollouts per action to estimate the gradient. As an example, SGD with one rollout per action and 25 iterations would require a total number of 100 rollouts, as it uses two two-sided finite differences. For BFGS the number of calls to (5.20) is varied in $\{50, 100, 200, 400, 800\}$ and each function call corresponds to $R$ rollouts. SGD only uses $R \in \{1, 4, 8\}$ which already shows good results, for BFGS in addition also $R = 16$ was tested.

Stochastic gradient descent shows very good results, being Pareto optimal to BFGS, in all but two configurations. It also shows a monotonic improvement of the optimization performance with an increasing number of iterations. For a high computational budget, it achieves a lower distance to optimum as the previous methods.

Interestingly, BFGS shows almost no improvement with a higher computational budget. A likely explanation is that the estimates of the action values with deterministic samples (5.20) are not as smooth as the true action value function $q^B$. An example of the sampled action values of the initial belief with two deterministic samples can be seen in Figure 5.7. One should note that the same holds true for

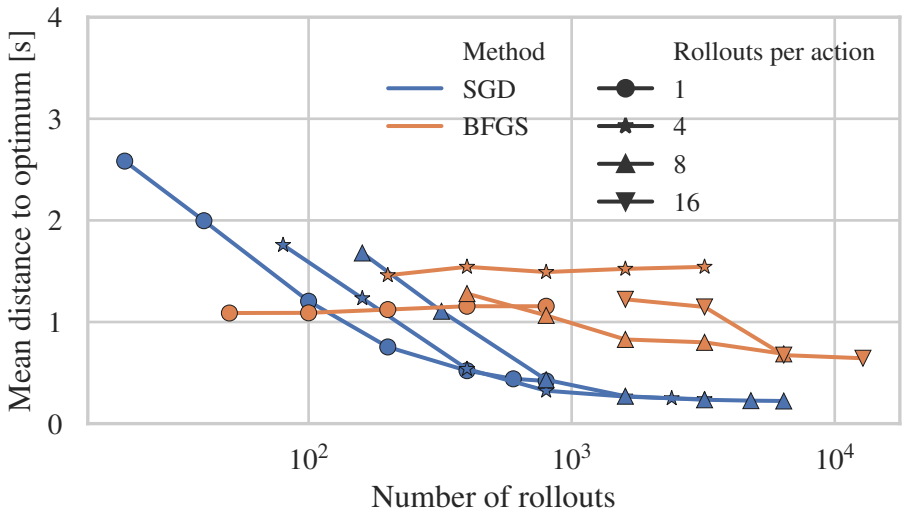
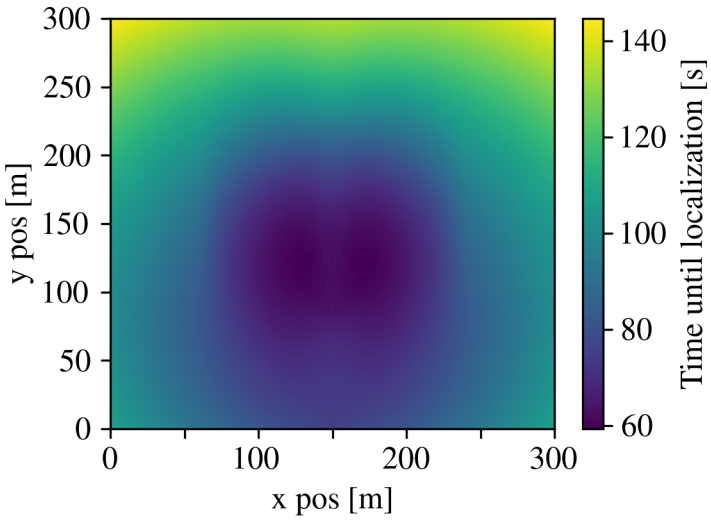



**Figure 5.6:** Optimization performance of the gradient based methods.

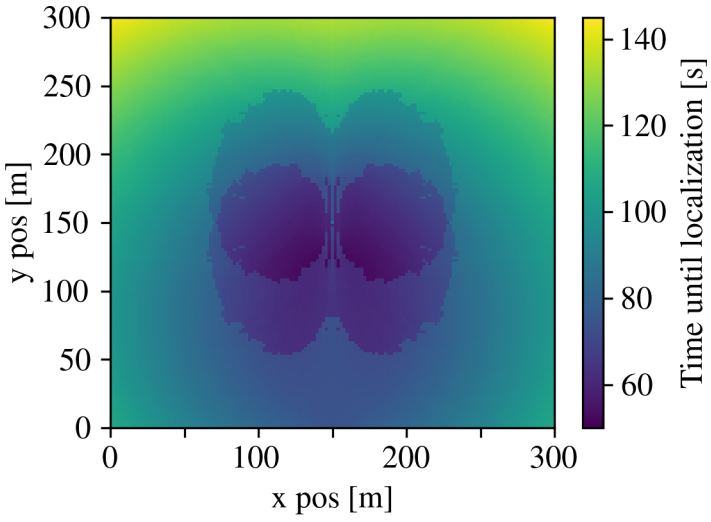
sampling the action value with a small number of PMC or CRN samples. However, stochastic gradient descent uses in each iteration a different sample path, therefore approximating gradient descent on the true action value function.

## Comparison

The number of rollouts the path planner can perform in practice will be limited by the computational power of the platform. Therefore, when we use the rollout path planner in a real sensor system, we will likely have a computational budget limiting the total number rollouts and we want to select the action selection algorithm and its configuration, such that the performance is the best possible. Figure 5.8 shows a subset of the results in a single figure for direct comparison. This figure gives the optimization performance achieved with a given number of rollouts and lets us compare the configurations with a common scale. With the exception of one quadrant search configuration, stochastic gradient descent is almost everywhere Pareto optimal to the other action selection algorithms. In addition, all algorithms that do not use the deterministic samples show a monotonous improvement and lead to better results when the number of rollouts is increased.




(a)




(b)

**Figure 5.7:** Action value approximation with deterministic samples. This figure shows (a) the approximately true action values $q^{B}(b_0, \mathbf{a})$ of the initial belief and (b) the approximation $\hat{q}^{B}(b_0, \mathbf{a}, \omega_{k[1:2]})$ computed with two deterministic samples.

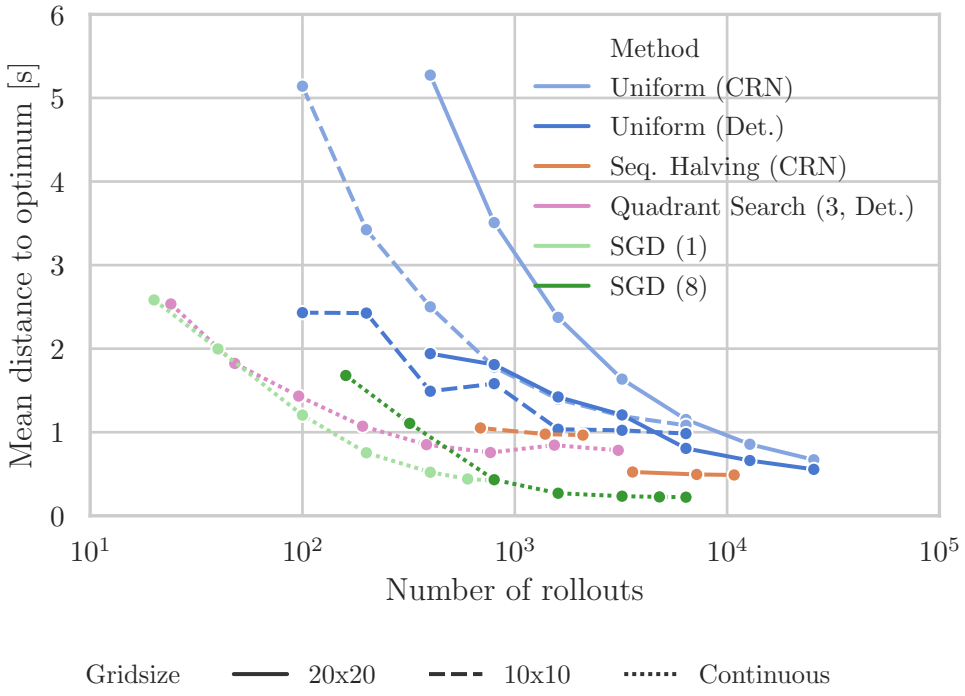



**Figure 5.8:** A selected subset of the plots in Figure 5.4, 5.5, and 5.6.

### 5.4.3 Localization performance

In the previous section, the different action selection algorithms were evaluated on a sampled set of beliefs, where for each belief a close approximation to the true action value function $q^B$ and the optimal action $\mathbf{a}_b^*$ had been computed. This quantified the optimization performance of an algorithm, which means how close it comes to selecting the minimum of $q^B$. However, in the end we are interested in how well the resulting rollout path planner works, which means how fast it actually localizes the target. We call this the *localization performance* and it is measured with the time until localization metric.

To evaluate the localization performance, we evaluated the same scenario with the same action selection algorithm configurations for 20000 MC runs. To determine the statistical significance of the results, the 95% confidence interval was computed for the mean time until localization of each configuration. Figure 5.9 shows the localization performance for the same methods as shown in Figure 5.8. ENTPP, the best performing path planner from the literature for Scenario 2 with medium

configuration, has an average time until localization of $67.25 \pm 0.71$ seconds (Table 4.2). Most configurations in Figure 5.9 lead to better results.

It can be seen that the configuration of Chapter 4, denoted by a star, is not Pareto optimal. With a high number of rollouts per action, the uniform allocation variant with deterministic samples achieves good results on the localization performance, even though it has a lower optimizing performance in Figure 5.8. However, these are also not Pareto optimal as sequential halving leads to comparable results but with a fewer number of samples.

In Figure 5.8, almost all configurations that are Pareto optimal in regard to the optimization performance belong to SGD. In comparison, Figure 5.9 shows that for a low computational budget SGD is dominated in the localization performance by the quadrant search method. In the localization performance SGD performs worse than would be expected from Figure 5.8. The worse performance of SGD is due to the initial belief. In this belief, the base policy selects an action far from the optimal action. The action of the base policy is used for the initialization of SGD and therefore the optimal solution cannot be reached if the number of iterations is small. This happens in every execution of the rollout with SGD because the initial belief occurs at the start of every execution. Uniform allocation, sequential halving, and quadrant search do not have this problem, as they do not require an initialization. However, for a high computational budget SGD receives results that are comparable with sequential halving, as one would expect from Figure 5.8.

Figure 5.10 shows two exemplary paths found by the rollout path planner with sequential halving and uniform allocation, and a high difference in the computational budget of 7200 to 400. This figure should give an intuition about how the resulting paths look like. The exact shape is dependent on the target position, the measurements, the configuration of the action selection algorithm, and random influences with the CRN and PMC sampling methods.

## Correlation with rollout performance

The previous section showed the optimization and localization performance of several action selection algorithms. While differences exist, for example in stochastic

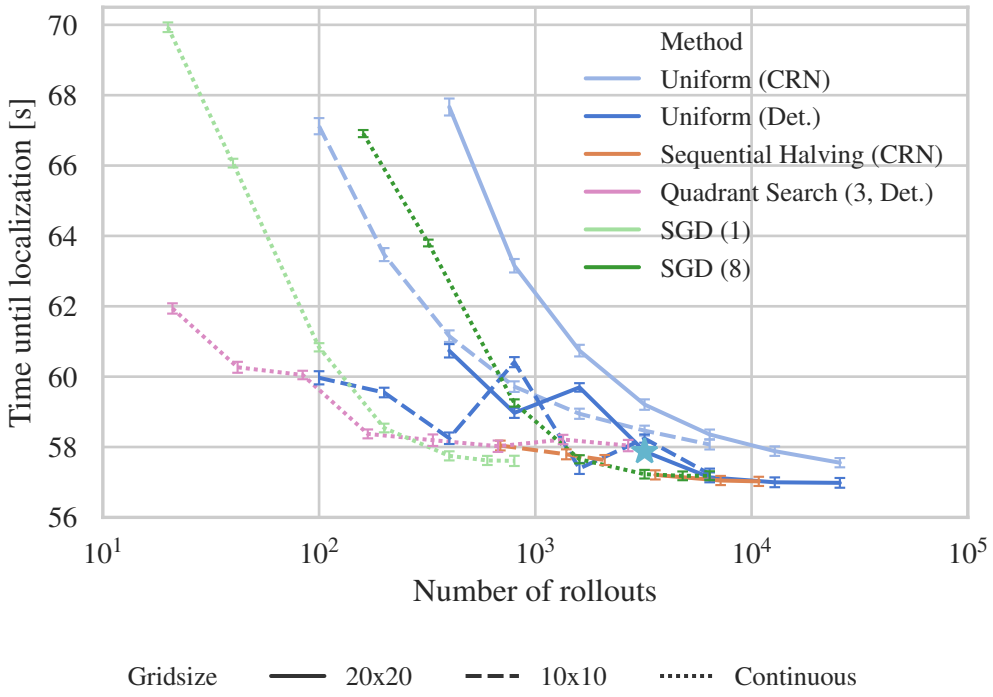



**Figure 5.9:** Localization performance for different number of rollouts, for a selected subset of methods. The star denotes the method used in Chapter 4. The error bars denote the 95% confidence interval.

gradient descent, the overall trend is similar. This is reassuring, as this indicates that by improving the optimization performance of the action selection algorithm, we can improve the performance of the resulting rollout algorithm. While one could expect that these values correlate, the base policy is only an approximation of future behaviour. Therefore, non-minimum actions might be actually optimal. The rollout improvement property only guarantees that if the action that minimizes $q^B$ is chosen, the resulting rollout path planner is at least as good as the base policy.

Figure 5.11 shows a correlation plot where the optimization performance is shown on the x-axis and the localization performance on the y-axis. It can be clearly seen that a correlation exists, with a Pearson correlation coefficient of $r^c = 0.775$. SGD and BFGS show several outliers from this correlation, where the time until localization is worse than expected. As previously mentioned, the worse performance of SGD is due to the initial belief, where the base policy does not provide a good initialization. Similarly, BFGS is initialized with the base policy and has the same

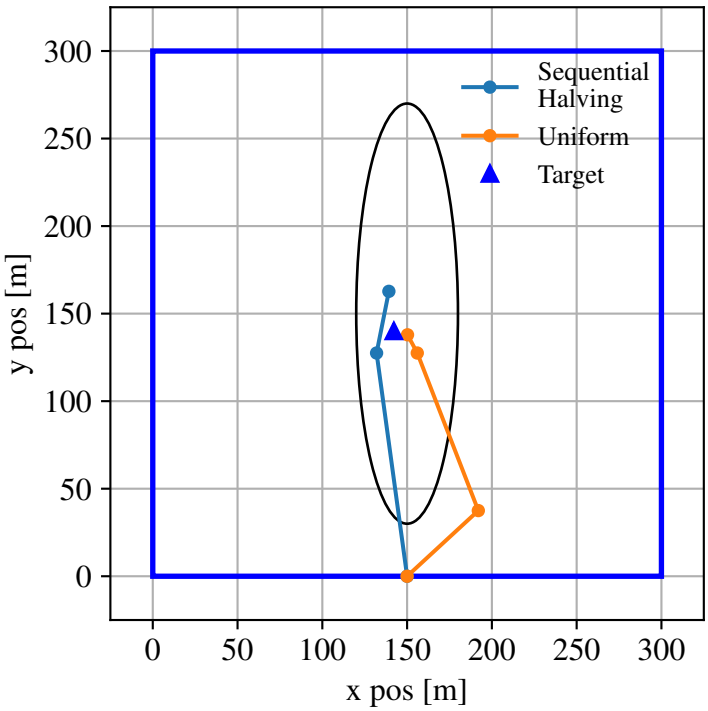



**Figure 5.10:** Exemplary paths for sequential halving (using CRNs with a $20 \times 20$ action grid and a computational budget of 7200) and uniform allocation, on the same grid using one rollout per action and plain Monte Carlo samples. In addition, the $3\sigma$ confidence ellipsoid of the initial belief is shown.

issue in the initial belief. The different frequency of occurrence for the initial belief explains the outliers in the correlation plot. The belief data set $\mathbb{B}$ consists of 25 different beliefs, which are created by 20 executions of the path planner. To avoid duplicate beliefs, the initial belief is only included once. However, this skews the data set, as the initial belief occurs relatively less often in $\mathbb{B}$ than in an actual execution of the rollout path planner. In comparison, each Monte Carlo run for computing the localization performance always starts with the initial belief. After correcting for this fact by weighting the initial belief by the factor 20 the correlation increases to $r^c = 0.965$.

## Scenario 3

Finally, we evaluate the localization performance of the action selection algorithms on a scenario different from that in the main part of the chapter. Figure 5.12 shows the geometry of this scenario, which we call Scenario 3. The target position in

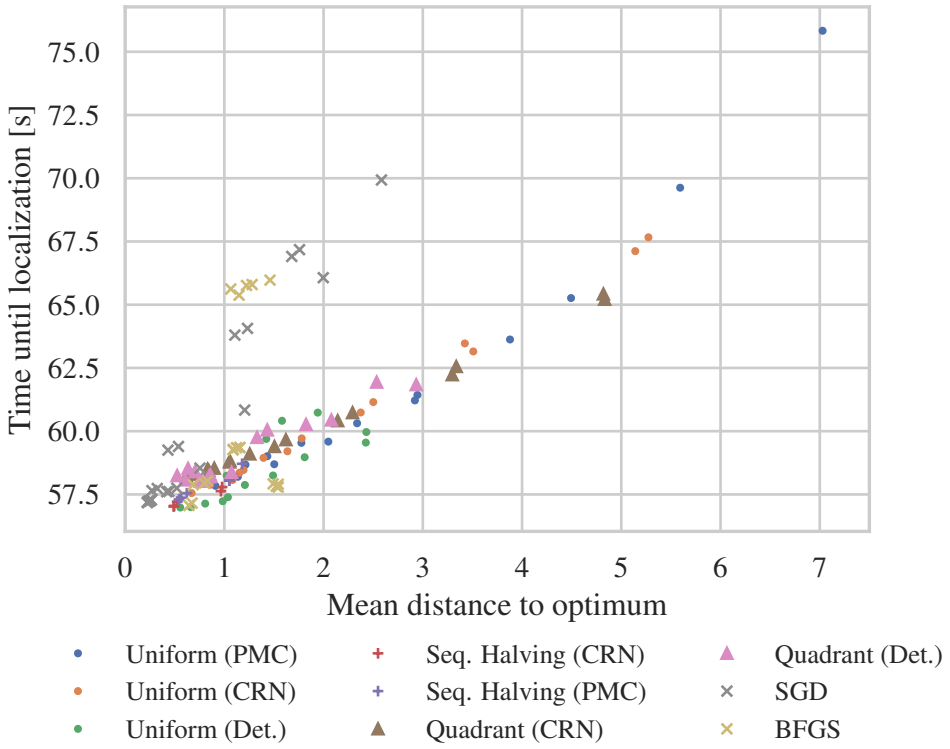



**Figure 5.11:** Correlation between the optimization performance and localization performance.

each Monte Carlo run is sampled uniformly in Cartesian space on a ring around the platform, with a distance of $30 - 300\,\mathrm{m}$. In this scenario, the platform takes an initial measurement directly at the beginning from its starting position, after which the initial belief is computed. Contrary to the previous localizer, the point estimate is made using the expected value of the belief, instead of the maximum a posteriori estimate. The remaining parameters are kept at the same values.

The intent of this scenario is to evaluate whether the relative localization performance of the different variants stays the same, when the distribution of target positions and the localizer change. Scenario 3 can be considered as a combination of Scenario 1 and 2, with targets at different distances but whose positions are randomly sampled according to the prior instead of being fixed. Its main difference

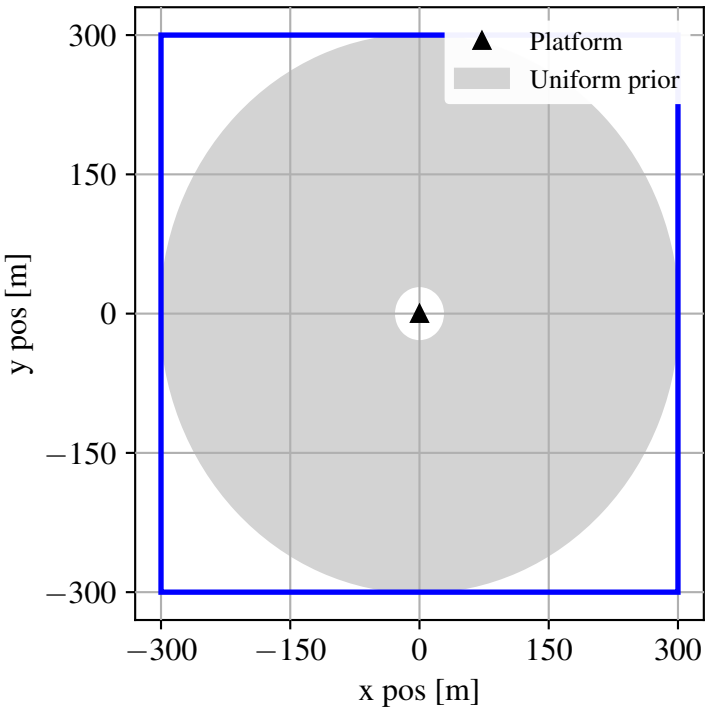



**Figure 5.12:** Scenario 3.

to both previous scenarios is a circular prior, different to the uniform prior in Scenario 1 and the Gaussian prior in Scenario 2. This simulates the case of a known maximum detection range.

Figure 5.13 shows the results of 20 000 Monte Carlo runs on this scenario. It shows the same methods as in Figure 5.8. As a reference also the entropy-based path planner (ENTPP), described in Chapter 4, was run on this scenario, resulting in an average time until localization of $98.22 \pm 0.2$ seconds. While there are differences in the absolute numbers, both figures show the same trends. Most configurations of the rollout path planner are better than ENTPP. Quadrant search is the best action selection algorithm for a low computational budget. The variants with deterministic samples show a non-monotonic improvement. Sequential halving is the best action selection algorithm for selecting from a finite action space. We also see a fast improvement with additional rollouts for stochastic gradient descent, which shows the best results for a high computational budget.

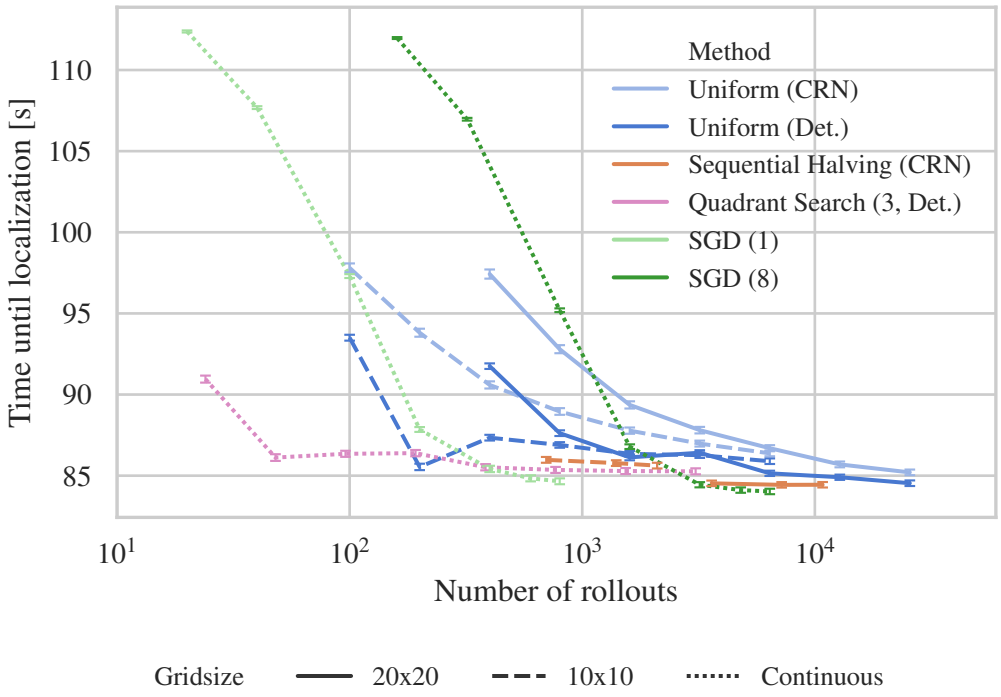



**Figure 5.13:** Localization performance dependent on the total number of rollouts for Scenario 3. The error bars denote the 95% confidence interval.

### 5.4.4 Sensitivity analysis

This section contains a sensitivity analysis of the results. We vary parameters of Scenario 2 and the rollout path planner and determine how robust our findings are to those changes.

In a first analysis, we evaluate the localization performance for several choices of the action grid resolution. Figure 5.14 shows the time until localization for different $X^a \times Y^a$ action grids, with $X^a = Y^a$ and 20 000 MC runs. The computational budget is set to a fixed number of $R = 16$ rollouts per action for uniform allocation. Sequential halving receives an equivalent computational budget of $N = 16 \cdot X^a \cdot Y^a$. The performance strongly improves until a $10 \times 10$ action grid, after which it shows diminishing returns for higher grid resolutions. For small grid resolutions, the performance shows alternating behaviour for even and odd numbers, as the corresponding grids contain different actions. For all action grid resolutions, common random numbers are preferable to plain Monte Carlo sampling. Sequential halving shows

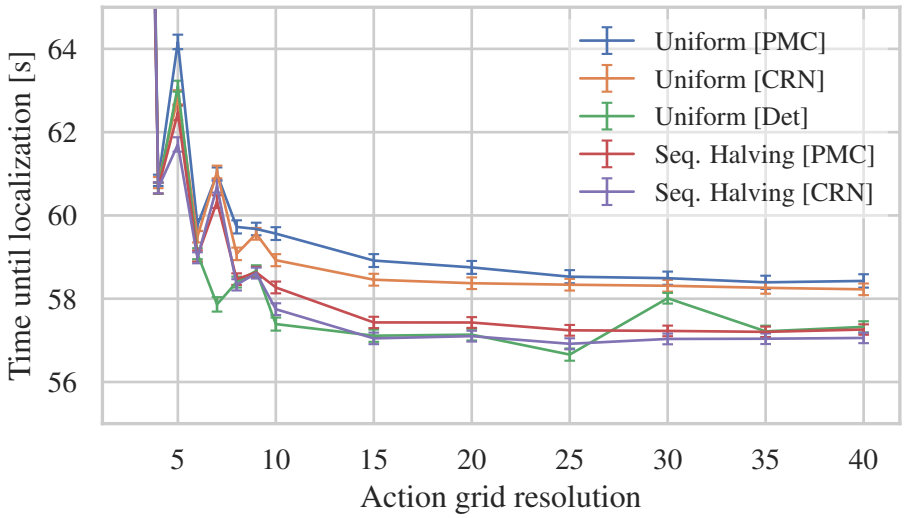



**Figure 5.14:** Variation of the action grid resolution. A fixed budget of 16 samples per action is chosen. The error bars denote the 95% confidence interval.

better results than uniform allocation for the same budget with both Monte Carlo sampling methods. Finally, deterministic sampling shows good results, comparable with sequential halving, but with a non-monotonic improvement for increasing the resolution of the action grid. This is similar to the results observed previously in Figure 5.9, where also a good performance for deterministic samples with a high computational budget, but with non-monotonic improvement was observed. It also mirrors the non-monotonic improvement in Figure 4.16, where the action grid resolution was increased for deterministic samples and $R = 8$ rollouts per action.

Figure 5.15 shows the time until localization for different measurement durations with 20000 MC runs. Different measurement durations lead to different trade-offs between taking a measurement and moving to another measurement location. We compare stochastic gradient descent with ten iterations and one rollout per action (40 rollouts in total), quadrant search with deterministic sampling, three iterations, and one rollout per action (24 rollouts in total) and sequential halving with a sampling budget of 3600. For comparison, also ENTPP is added. Figure 5.15 shows that the advantage of quadrant search to stochastic gradient descent for a small computational budget is robust over different measurement durations. For a small computational budget, ENTPP performs approximately as well as the

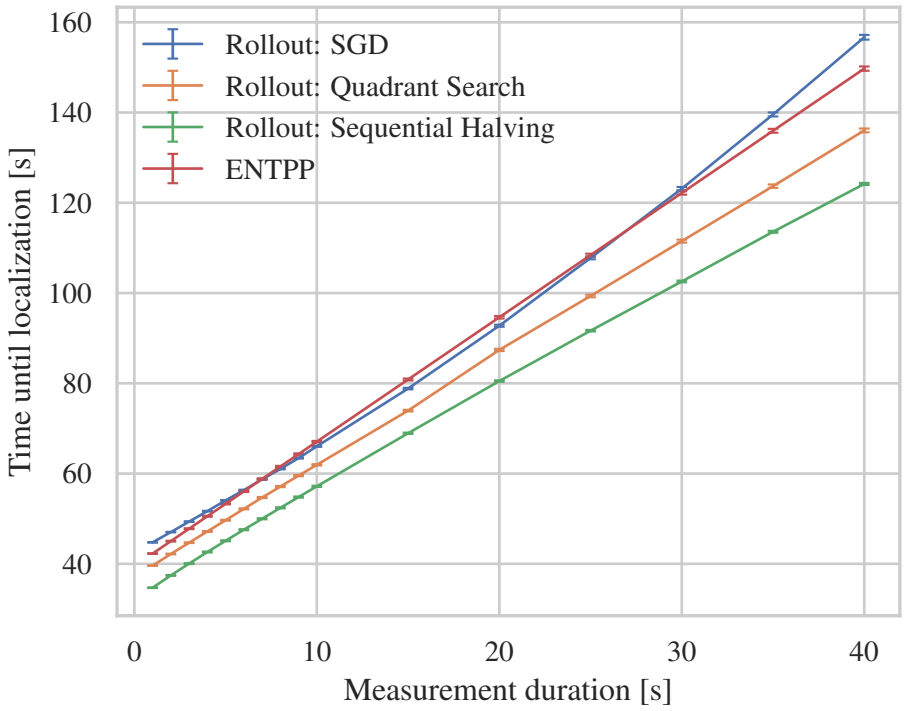



**Figure 5.15:** Variation of the measurement duration $t^m$. Stochastic gradient descent and quadrant search use a small computational budget with 40 and 24 rollouts. Sequential halving has a computational budget of 3600 and a $20 \times 20$ grid. The error bars denote the 95% confidence interval.

rollout path planner with stochastic gradient descent as action selection algorithm. However, quadrant search and sequential halving with a higher budget outperform ENTPP for all variations of the measurement duration.

## 5.5 Conclusion

As a first conclusion, we note that using common random numbers should be preferred to plain Monte Carlo when comparing action values. Deterministic sampling of the belief improves the performance of some action selection algorithms, however, not necessarily. A likely explanation is that it improves the results in algorithms where the search evaluates actions that are not close to each other. This is the case in uniform allocation, due to the grid, and in quadrant search because the initial nine actions cover the whole extent of the action space. On the other hand, it likely leads to worse results in algorithms where the search is local as in BFGS,

because it might produce local minima. Different to PMC and CRN, deterministic samples do not produce a monotonic improvement in optimization and localization performance.

For the given problem, there were two sources of uncertainty: uncertainty in the current belief and uncertainty in the future measurements. The deterministic samples are only used for the first source of uncertainty. Using the expected value for future measurements implicitly assumes that it is more important to capture the uncertainty in the current state estimation than the uncertainty in the future measurements. This seems to work for the given problem, however, does not necessarily need to generalize to other problems where the uncertainty in the future measurements might have a higher influence.

Sequential halving is an effective action selection algorithm with discrete or discretized action spaces. As it uses Monte Carlo samples, it does not need to make any assumptions about the uncertainty. An additional advantage is that given the action space, the only hyperparameter to be determined is the computational budget. A disadvantage is that it requires at least a computational budget of $N = |\mathbb{A}_k| \cdot \lceil \log_2 |\mathbb{A}_k| \rceil$.

Quadrant search works well and is worth considering if the computational budget is low. As it focuses on a single quadrant after evaluating only nine actions, it has the possibility to focus the search on the wrong quadrant, but in the considered problem this seems to be not an issue.

Stochastic gradient descent shows the strongest results when considering its optimization performance on the belief set $\mathbb{B}$. For a small computational budget, the localization performance of the resulting rollout path planner is worse than expected, due to worse initialization in the initial belief. However, with a high computational budget, the localization performance is also good, and its strong optimization performance makes it worth considering. In this work, we only evaluated a single step size schedule. This parameter has likely a significant and potentially problem dependent influence on the performance. Therefore, a higher performance might be possible, however, at the cost of more intensive hyperparameter tuning.

An important consideration is the inclusion of constraints on the sensor state space and thereby action space. It is easy to incorporate constraints into uniform allocation and sequential halving, as actions could simply be excluded from the discrete action space. The implementations of stochastic gradient descent and BFGS assumed an unconstrained sensor state space of $\mathcal{X}^s = \mathbb{R}^2$. A possible way to add a constraint to these methods, would be a projection into the action space if those algorithms evaluate an infeasible action or a method as L-BFGS-B [Byrd et al., 1995] which allows for constraints. An extension of the quadrant search to a non-rectangular action space would not be straightforward.

The main contributions of this chapter are the following:

1. It formulates a model for adaptive action evaluation in the policy rollout in form of an action selection algorithm. This models the action decision as a stochastic function minimization problem and allows to use existing minimization methods on this problem, of which several were never used in the policy rollout context before. Especially, some of those methods explicitly support a continuous action space, while the literature previously only considered discrete action spaces.

2. It analyses the performance of the resulting rollout path planner and how it correlates with the optimization performance. It is shown that there is a clear correlation, further motivating the optimization of the action selection step.

3. It contains a sensitivity analysis of the results and shows that they hold also for different parameters than originally used in the scenarios.

As a result of this chapter, the trade-offs of different action selection algorithms are quantified. This allows us to select an algorithm for the actual implementation of the rollout path planner in the experiments. The implementation in the experiments is based on sequential halving with common random numbers. This method had shown a robust performance and also works well with constraints on the action space that are present in the experimental area. The next chapter describes further

adaptions to the rollout path planner due to limitations in a real sensor system and shows experimental results.

# Chapter 6

# Experimental Sensor System

The previous two chapters introduced a path planner for the emitter localization problem. This path planner shows good performance in simulations. However, the more valuable test for an algorithm is to show that it works in a real system. This chapter describes an experimental sensor system that is controlled by the rollout path planner and shows experimental results.

Section 6.1 reviews similar experimental systems described in the literature. The systems are categorized into those that use the strength of the received signal and those that use bearing measurements. A focus in the review is put on whether sensor path planning is used in the experimental system.

During the development of the sensor system, several challenges had to be overcome. These required changes to the path planner, discussed in Section 6.2. First, the used direction finding sensor does not produce reliable measurements if it is too close to the target position, because of the high elevation of the received signal. As solution, sensing actions too close to the confidence region of the belief are removed from the action space. Second, the base policy of Chapter 4 is not feasible any more, due to the constrained action space. As solution, a new base policy is used. Finally, occasionally outliers occur. Therefore, the received power as a function of the UAV heading is compared with the antenna pattern and the measurement is rejected if those are too different.

The sensor system is described in Section 6.3. It consists of a single UAV with a directional antenna, controlled by a ground station. In total six flights on three

days were made to evaluate the sensor system. In eleven localization attempts, the UAV was controlled by the rollout path planner. The results are given in Section 6.4. Of those eleven localization attempts, in seven the target was inside the indicated confidence region, in two cases close to the point estimate but not in the confidence region, and in two localization attempts outliers occurred.

In addition, Section 6.4 analyses the measurements and compares the time prediction model from the path planner with the results from the experiments.

## 6.1 Comparison with existing work

In this section we review mobile sensor systems for RF emitter localization that are described in the literature. These sensor systems can in general be categorized into those measuring the signal strength, also called *received signal strength indicator* (RSSI) and those that compute a bearing towards the emitter. An RSSI-based system typically uses a known reference signal strength of the emitter to compute an estimated range.

Except otherwise noted, the platforms are multicopter UAVs. Sensor systems are included in the review whether they perform sensor path planning or not. The main intent is giving an overview over the solution space of the problem of emitter localization with a mobile sensor system. From those systems that perform sensor path planning, it turns out that the majority fields myopic path planners.

### 6.1.1 RSSI-based sensor systems

RSSI-based sensor systems [Nguyen et al., 2019, 2020; Körner et al., 2010; Vrba et al., 2019] use the fading of signal power over distance to compute a range estimate for the emitter. As the platform does not have to remain stationary during the measurement process, the sensor typically produces measurements at regular time intervals.

The sensor system described by [Nguyen et al., 2019] localizes multiple radio tags using signal characteristics to associate the measurements. The localization is done by a particle filter that uses a reference signal strength to compute a range estimate. Sensor path planning is performed by predicting the influence of a single

action and computing the expected Rényi divergence using Monte Carlo sampling. In the field experiments, a planning horizon of one time step (5 s) is used.

The work in [Nguyen et al., 2020], also uses a particle filter to localize multiple targets. The intent is to create a computationally lightweight path planner that can run embedded on the sensor payload. The path planner works by first selecting the target with the smallest uncertainty above a threshold and then testing three direct paths towards a circle around this target, two tangents and the direct line towards the target position. If, based on the particle filter belief, the probability of one of those lines coming close to another target is sufficiently small, this line is selected and the UAV moves along that line to a new position. Otherwise the heading change that lets the UAV come closest towards the selected target, while having a sufficiently small probability of not coming too close to other targets, is selected. The constraint of the minimal distance is motivated by not disturbing the animals that are wearing the RF emitter that are localized.

The sensor system described in [Körner et al., 2010] is also RSSI-based and uses a particle filter to track multiple targets, that emit signals at the same frequency. The sensor only returns the strength of the strongest signal, but it was shown in simulations that tracks can be correctly updated due to the spatial separation of the targets. The field tests contained only a single target. While the sensor system was intended for a fixed wing UAV, the field test was performed with the system mounted on a car. No sensor path planning was performed in this work.

In [Vrba et al., 2019] a sensor system consisting of three UAVs was described, which localizes Bluetooth beacons via RSSI. Contrary to the previous systems, this one uses an extended Kalman filter (EKF) to localize the targets. A separate EKF belief was instantiated for each unique beacon ID. Two beacons were localized in a field test, with the UAVs flying on a predefined trajectory. A path planner was developed based on the idea to move the centre of the UAV formation on top of the estimated target position, but only tested in simulations.

### 6.1.2 Bearing-based sensor systems

Bearing-based sensor systems are the major alternative to RSSI-based systems. These systems measure the target bearing, often by rotation of a directional antenna. Bearing measurements have the advantage that a target can be localized even at further distances and with fewer measurements. However, a disadvantage compared to RSSI-based sensor systems is that bearing measurements require more complex signal processing, while measurements of the signal strength can be directly read from the receiver. Bearing measurements also require the orientation of the UAV to be known, while with signal strength measurements the position is sufficient.

In [Cliff et al., 2015] a sensor system is described that consists of a directional antenna, mounted on a UAV. By rotating the UAV the target bearing is measured and the target localized with a grid-based Bayes filter. Path planning is done myopically by considering the next measurement locations and computing for each the entropy of the belief updated with the expected measurement. This path planner was previously discussed in Chapter 4, where it is referred to as ENTPP.

[Isaacs et al., 2014] uses a similar sensor system consisting of a UAV mounted directional antenna to localize a moving target with a particle filter. The focus of this work is on the low level control of the UAV position and orientation, by ensuring a steady rotation at constant altitude with a PID controller.

The work in [Vonehr et al., 2016] discusses also a rotating UAV with a directional antenna. It focuses on the signal processing aspect of the sensor system. Only the measurement error is analysed and the target is not localized. Therefore, also no path planning is considered in this paper.

The sensor systems described above require an explicit rotation of the UAV. In comparison, the work presented in [Venkateswaran et al., 2013] is based on a single-wing monocopter which similar to a maple seed rotates continuously during its normal flight. On the UAV a directional antenna is attached to determine bearing measurements. The target is not localized, instead the goal of the system is to fly directly above the target. This is done by flying in the direction of the bearing measurement.

A sensor system that does not require a rotation at all is presented in [Dressel and Kochenderfer, 2018]. In addition to a directional antenna, the UAV also carries an omnidirectional antenna. By comparing the received power from the directional antenna with the omnidirectional one, the gain of the directional antenna towards the target bearing can be determined. However, the resulting bearing measurement is ambiguous, as the antenna might have the same gain in multiple directions. Target localization is performed with a grid-based Bayes filter using a likelihood function that incorporates these ambiguities. A myopic path planner is used that plans the next sensing action based on the expected entropy of the updated belief.

A robotic system to monitor carp in a lake is described in [Tokekar et al., 2011, 2013; Vander Hook et al., 2014, 2015]. The sensor system uses a directional loop antenna, which is rotated by a motor. As platforms, either robotic boats or wheeled robots are used, dependent on whether the lake is frozen or not. The targets are localized using an extended Kalman filter. Several path planners have been developed for this system. In [Tokekar et al., 2011] three path planners are proposed of which two are tested experimentally. The work of [Vander Hook et al., 2015] extends the path planning to multiple robotic boats and optimizes the time until localization. In chapter 4 this path planner was referred to as SOPT and used as a comparison algorithm. [Vander Hook et al., 2014] presented a myopic path planner, similar to the base policy used in this thesis. Section 4.1 contains more information about these path planners.

A ground-based robot with a directional antenna, rotated by a motor is also used in [Graefenstein et al., 2009] to localize targets with a particle filter. The system is tested for bearing measurements with a stationary robot and with a moving robot. As expected, the bearing measurements are worse during movement of the robot, but still useful. The robot follows an arbitrary, not explicitly optimized path.

The work in [Vrba et al., 2019] was already discussed in the context of RSSI and also presents results from a UAV equipped with a directional antenna. Different from the other bearing-based sensor systems on UAVs, the UAV payload contains a motor to rotate the antenna without the UAV rotating. The bearing measurements

are taken from predefined measurement locations and the localization is done using a weighted least squares estimator.

### 6.1.3 Contributions of this chapter

As the above literature review shows, the localization of RF emitters with mobile, unmanned sensor systems has attracted considerable interest in the literature. The main novelty of the sensor system described in this chapter is in the path planner.

While path planning has been considered in the literature, the majority of works only field a myopic path planner. Only few works [Tokekar et al., 2011; Vander Hook et al., 2015] consider a nonmyopic planner. However, [Vander Hook et al., 2015] is limited to two future measurement locations and [Tokekar et al., 2011] does not consider the time required for a measurement. Most of the works use an information-driven cost function, with only [Vander Hook et al., 2015] optimizing the time until localization. Only [Tokekar et al., 2011] describes a path planner for a bearing-based sensor system that takes into account different outcomes of the measurements. However, as described in Section 4.1, it does not use the full mobility of the system.

In contrast, the sensor system described in this chapter uses the path planner described in Chapter 4 and 5, which performs a nonmyopic planning over the full localization process, taking into account the uncertainty of the estimate and the future measurements.

## 6.2 Changes to the path planner

This section discusses the required changes of the rollout path planner to make it work in a real sensor system. The first section discusses constraints on the action space to ensure measurement locations are not too close to the target position. In the second section a new base policy for handling the constrained action space is introduced. Finally, the third section discusses a detection method for measurement outliers.

### 6.2.1 Constraints on the action space

A restriction of the sensor is that the antenna pattern is also elevation dependent. Therefore, a measurement taken close to the target is extremely unreliable.

We consider this restriction in the path planner by limiting $\mathbb{A}_k$ to actions sufficiently distant from probable target positions. These positions are given by the *confidence region* of the belief. This region is similar to the confidence ellipsoid, but for a grid-based Bayes filter it is not necessarily an ellipsoid. We define the confidence region as the convex hull of the grid cells containing 95% of the probability mass. For this, the grid cells of the localizer are sorted with descending probability mass, and the cells corresponding to the top 95% percentile constitute the confidence region. From the positions of those cells the convex hull $\mathcal{C}^{95}(b_k^t)$ is computed. As a second step, the action space is reduced to those actions that keep at least a minimal distance from the confidence region. The action space at decision step $k$ is then

$$\mathbb{A}_k = \left\{ \mathbf{a} \in \mathbb{A} : d(\mathbf{a}, \mathcal{C}^{95}(b_k)) > \underline{r} \right\} \tag{6.1}$$

where $\underline{r}$ is the minimal distance and $d(a, \mathcal{C}^{95}(b_k))$ computes the minimal distance between action $\mathbf{a}$ and convex hull $\mathcal{C}^{95}(b_k)$. $\mathbb{A}$ is the original action space, which is a discretization of the scenario area. An example of the confidence region and a reduced action space can be seen in Figure 6.12. If $\mathbb{A}_k$ is empty or contains only one action, the minimal distance $\underline{r}$ is reduced until there are at least two actions. However, this situation never occurred during the experiments.

### 6.2.2 Base policy

The base policy defines a baseline behaviour for approximating the future movement of the UAV. In Chapter 4 and 5, the path planner uses a base policy that does not consider any constraints on the action space. However, the small size of the experimental area and the requirement of a minimal distance to the target constrain the action space significantly. In this constrained setting, the base policy from Chapter 4 leads to too optimistic judgements of some actions, as the rollout would simulate impossible future action sequences.

Therefore, we utilize a base policy that uses the same constrained action space (6.1) and greedily selects sensing actions based on the determinant of the Fisher information. The base policy uses the point estimate $\tilde{\mathbf{x}}_k^t$ and covariance $\tilde{\mathbf{P}}_k^t$ of the belief, and selects actions according to

$$\pi^B(b_k) = \underset{\mathbf{a}\in\mathbb{A}_k}{\operatorname{argmax}} \det\left(\left(\tilde{\mathbf{P}}_k^t\right)^{-1} + \mathbf{J}(\tilde{\mathbf{x}}_k^t, \mathbf{a})\right) \tag{6.2}$$

where $\mathbf{J}$ is given by (3.78). It should be noted that this base policy is only feasible because sensing actions too close to the confidence region are removed from the action space. Otherwise the most informative action would always be the one that lies directly over the point estimate, because the Fisher information (3.80) is proportional to $1/r^2$ with $r$ being the distance between measurement location and target.

### 6.2.3 Detection of measurement outliers

Even for measurements taken at a sufficient distance, outliers occur. In most of those cases the received power as a function of the UAV heading was different than the antenna pattern. Therefore, we use a cosine similarity criterion to detect measurements that differ from the pattern. This criterion is applied after the direction finder has computed a measurement $z_k$.

At measurement step $k$, the *measured power vector* $\mathbf{m}_k \in \mathbb{R}^{180}$ is defined as the vector that contains in $2^\circ$ bins the average power that was received from each direction. With a perfect measurement, $\mathbf{m}_k$ would correspond to a shifted version of the antenna pattern $\boldsymbol{\psi} \in \mathbb{R}^{180}$, shown in Figure 6.5. The shifted antenna pattern $\boldsymbol{\psi}(z_k) \in \mathbb{R}^{180}$ is the antenna pattern, shifted according to the measured bearing $z_k$. With a perfect measurement, these $\boldsymbol{\psi}(z_k)$ and $\mathbf{m}_k$ would be identical. Then, for the measurement $z_k$ returned by the direction finder, the *cosine similarity*

$$\zeta(z_k) = \frac{\mathbf{m}_k}{\|\mathbf{m}_k\|_2} \cdot \frac{\boldsymbol{\psi}(z_k)}{\|\boldsymbol{\psi}(z_k)\|_2} \tag{6.3}$$

is a value between -1 and 1 that indicates the similarity between the measured power vector and the antenna pattern. A measurement is accepted if this value is above the *cosine similarity threshold* $\underline{\zeta}$, and otherwise a new measurement is taken at the same position. We call this decision criterion *cosine similarity criterion* because the inner product can be interpreted as the cosine of the angle between those two vectors. While the underlying direction finder [Krestel et al., 2019] potentially returns multiple measurements, the cosine similarity criterion makes the assumption that only a single target is present.

Similar criteria for the measurement quality have been used in [Graefenstein et al., 2009; Cliff et al., 2015]. In [Graefenstein et al., 2009] measurements below a threshold are rejected, as done in this thesis. In [Cliff et al., 2015] the standard deviation of a measurement is estimated based on the quality criterion.

## 6.3 Experimental setup

The sensor system consists of three parts, the UAV, payload, and ground station. As UAV the *AirRobot*[1] *AR200* UAV is used. A picture of the UAV with equipped payload can be seen in Figure 6.1. This UAV can carry a payload up to 3 kg and allows for automatic control with an API. The payload contains the actual sensor and a computer to perform the direction finding. The ground station consists of the control station of the UAV, as well as a laptop that runs the path planner. The control station is connected to the UAV by direct radio and the ground computer to the payload computer via LTE over the cellular network.

The target is an RF emitter, which sends an unmodulated sine wave at 1984 MHz. This frequency was chosen due to an existing license to send on this frequency. The emitter consists of a *Raspberry Pi Zero* and a *LimeSDR Mini* software defined radio, and can be seen in Figure 6.2.

After placing the target in the experimental area, its position is measured using the GPS of the UAV by placing it above the target. The GPS of the UAV is used to avoid any systematic errors between two different GPS receivers. The resulting

[1] `https://www.airrobot.de/`

**Figure 6.1:** Platform with mounted payload. (Picture: Markus Krestel)

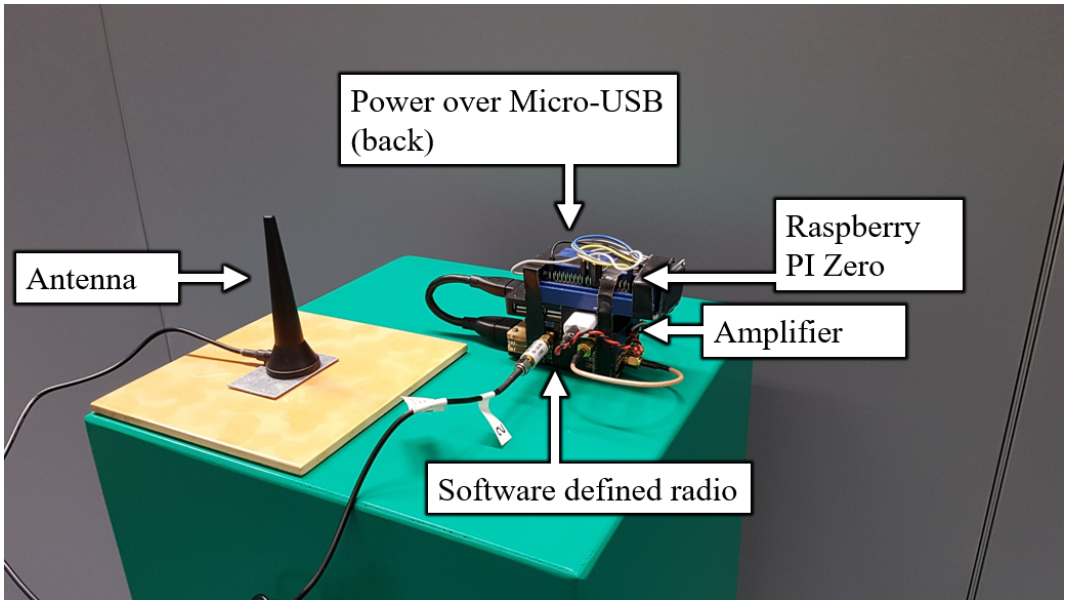



**Figure 6.2:** The RF emitter used in the experiments.

position is referred to as *true target position*. In the literature this value is also known as *ground truth*.

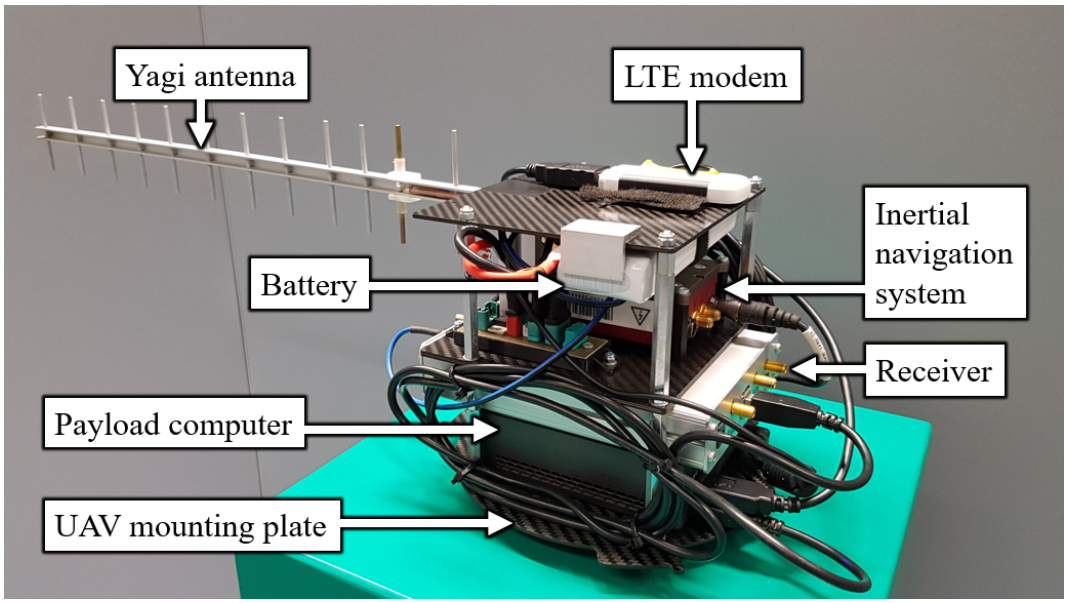



**Figure 6.3:** Payload.

### 6.3.1 Hardware description

The payload attached to the UAV consists of a Yagi antenna, a *USRP B210* receiver, an *Ellipse-D* inertial navigation system (INS), and a *Huawei e3372h* LTE modem for communication. As *payload computer* we use a *Gigabyte Brix 8550*. Figure 6.3 shows an image of the payload and Figure 6.4 a block diagram. The total weight of the payload amounts to 2.061 kg. The INS is connected to two GPS antennas which are fixed on the arms of the UAV with 1.20 m separation. This allows the INS to compute the heading of the UAV.

The pattern of the Yagi antenna is illustrated in Figure 6.5. The antenna pattern needs to be measured with the payload mounted on the UAV, as it is potentially influenced by the UAV. The measurement of the antenna pattern was performed outside as can be seen in Figure 6.6. The UAV was placed on a hill and a signal generator was directed upwards to minimize interference. This also measures the azimuth pattern at a nonzero elevation as is the case in the experiments. The UAV was mounted on a *PT-3002* pan/tilt head and the signal power was measured in steps of 2°.

The ground station is shown in Figure 6.7. It consists of the UAV control station and a laptop, which we call *ground computer*. The UAV is controlled from the

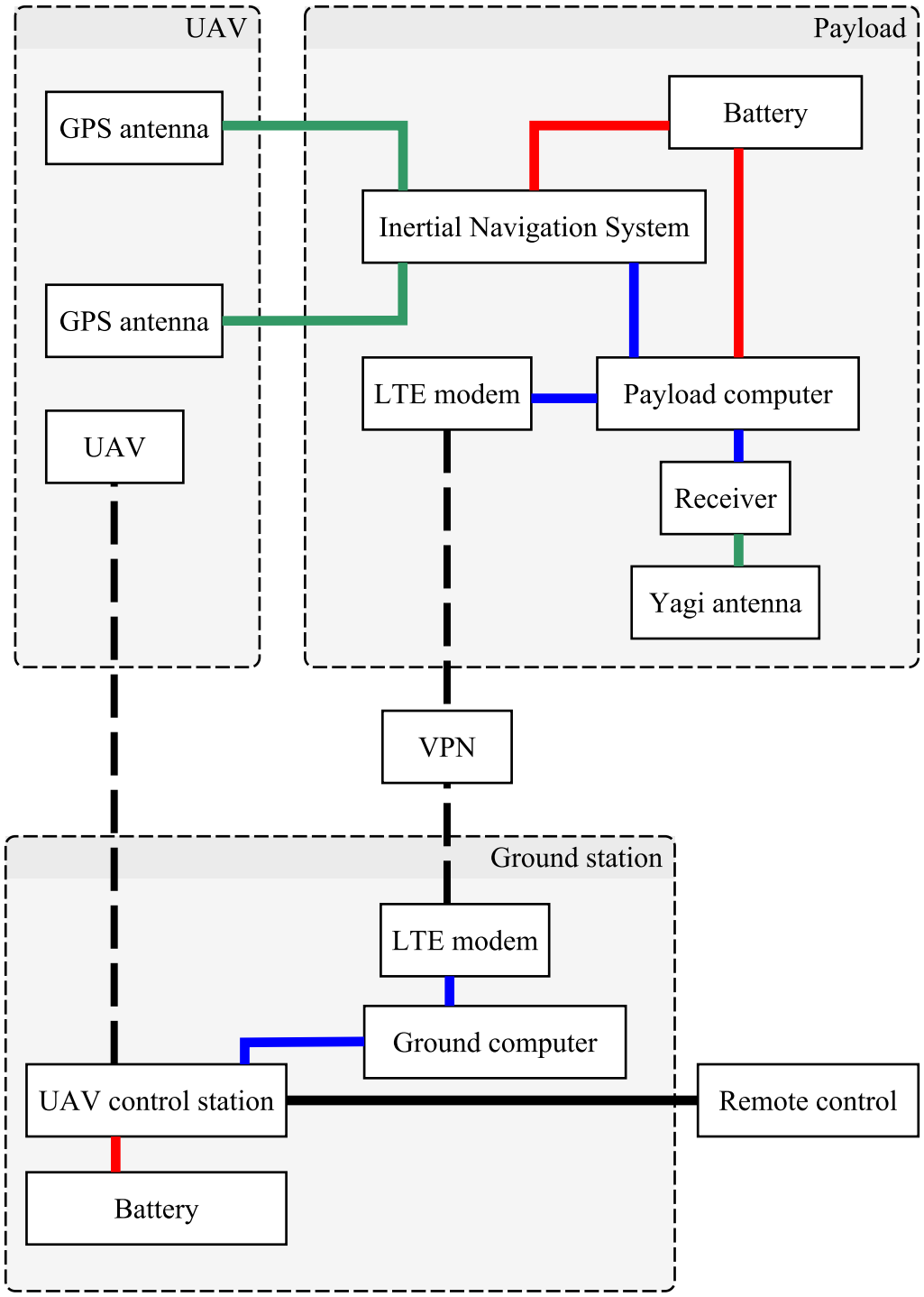



**Figure 6.4:** Block diagram of the hardware. Red indicates a power line, blue a USB connection and green an antenna cable. The dashed lines are radio connections.

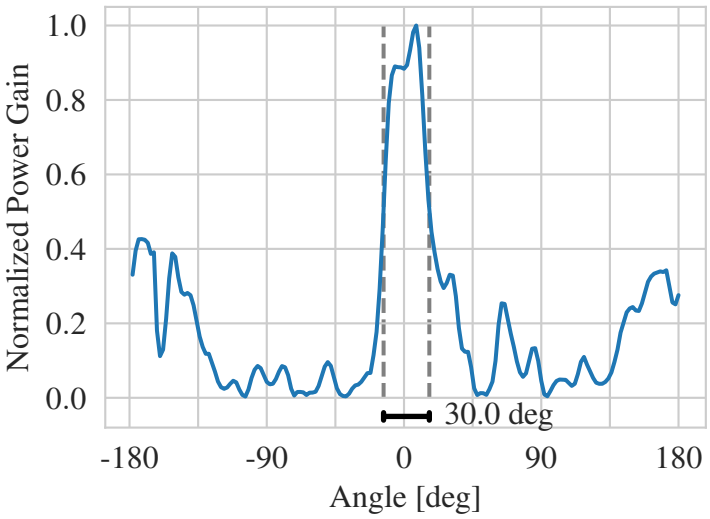



**Figure 6.5:** Antenna pattern, mounted on the UAV.

**Figure 6.6:** Measurement of the antenna pattern.

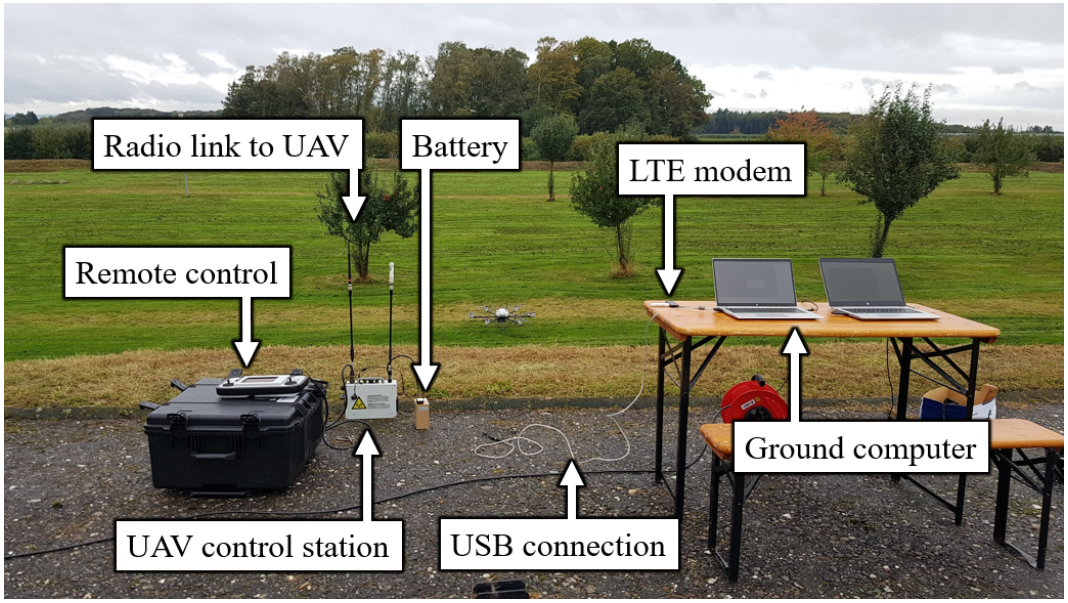



**Figure 6.7:** The ground station of the sensor system. It is placed on the same hill as visible on Figure 6.6.

control station. The path planner runs on the ground computer and communicates with the control station over a USB connection. In addition, the ground computer is connected to the payload via an LTE connection. The ground computer is equipped with a 2.1 GHz, 6 core *AMD Ryzen 5 PRO 4650U* CPU and 16 GB RAM.

The used consumer-grade LTE contract did not support fixed IP addresses and the option to directly create a point-to-point connection via LTE. Therefore, the payload and ground station connect to a virtual private network (VPN), which is accessible from the internet. In the VPN both systems have fixed IP addresses and can communicate with each other.

In addition to the automatic control by the path planner, the UAV is also controllable by a remote control. For legal reasons, a human pilot needs to be able to interrupt the planning process at each instant. In addition, the take-off and landing phases were performed by the human pilot.

From the terms established before, the sensor consists of the Yagi antenna, receiver, INS, and payload computer. These components are responsible to create the bearing measurements. The UAV corresponds to the platform and the sensor system is the joint system of UAV, payload, and ground station.

### 6.3.2 Software description

The software of the sensor system is based on the *robot operating system* (ROS), version 16.04 *Kinetic Kame*. ROS is a middleware that enables the communication between different processes, called *nodes*, by messages ordered by *topics* [Quigley et al., 2009]. A separate instance of ROS is run on the payload computer and laptop. The *FKIE multimaster* package[2] synchronizes these instances. The *FKIE multimaster* is configured by a list of topics and services and replicates them on the other system. The communication for this takes place over the LTE connection. For logging purposes the messages from all topics are written on disk, in a file called *rosbag*.

On the payload runs a node controlling the receiver, as well as a node performing the direction finding. The ground station initiates the rotation of the UAV and starts the measurement process by a service call to the direction finding node. This node activates the receiver node which publishes samples of the received power at regular intervals. After a full rotation the direction finding node computes a bearing measurement and publishes this on the corresponding topic.

The direction finding algorithm was implemented in prior work [Krestel, 2018; Krestel et al., 2019]. The algorithm performs direction finding with multiple targets by computing an optimal reconstruction

$$\mathbf{r}_k^* = \underset{\substack{\mathbf{r}\in\mathbb{R}^{720}\\ \mathbf{r}\geq\mathbf{0}}}{\operatorname{argmin}} \|\mathbf{A}\mathbf{r}-\mathbf{m}_k\|_2 + \lambda\|\mathbf{r}\|_1 \tag{6.4}$$

of the signal. The elementwise nonnegative vector $\mathbf{r}\in\mathbb{R}^{720}$ denotes the estimated average power from each direction, discretized in $0.5^\circ$ bins. Each row in the matrix $\mathbf{A}\in\mathbb{R}^{180\times720}$ consists of the antenna pattern, shifted by $0.5^\circ$ for each row. The measured power vector $\mathbf{m}_k\in\mathbb{R}^{180}$ stores the measured power from each direction, discretized in $2^\circ$ bins. The expression $\mathbf{A}\mathbf{r}$ computes the expected power vector if for each angle $i\cdot 0.5^\circ$ a signal with strength $\mathbf{r}_i$ is incoming. The parameter $\lambda = 0.3$ penalizes a too large vector $\mathbf{r}$. Effectively, this algorithm reconstructs the measured

[2] https://github.com/fkie/multimaster_fkie

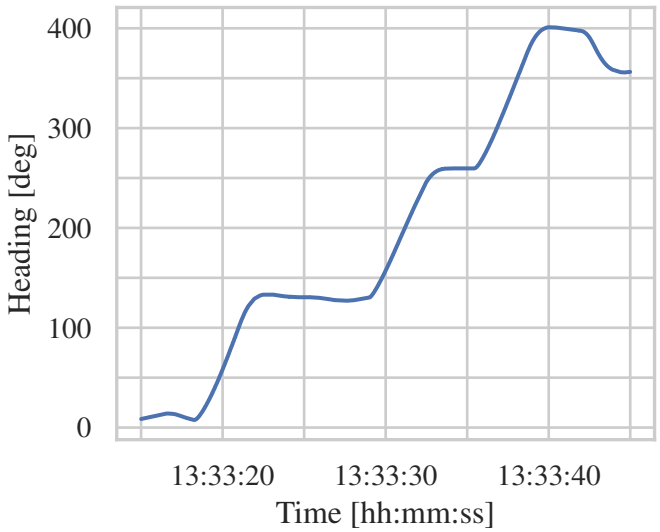



**Figure 6.8:** Rotation during the measurement process. The last rotation from 400 to 350 degrees is part of the movement to the next measurement location.

power vector by a superposition of the antenna pattern shifted to different measurement locations. The minimization problem (6.4) is also known as *basis pursuit denoising problem* [Krestel et al., 2019]. As only a single target is present in the experiments, only the strongest measurement

$$z_k = i \cdot 0.5^\circ \quad \text{with} \quad i = \underset{1 \leq i \leq 720}{\operatorname{argmin}} \; (\mathbf{r}_k^*)_i \tag{6.5}$$

is returned.

The rollout path planner is executed on the ground computer as a Java application. The implementation uses multi-threading to compute the rollouts of a single sequential halving iteration in parallel. In the experiments, six threads were used. A ROS node exists as a bridge that subscribes to the topic containing the bearing measurements and forwards them to the path planner. The communication between the path planner in Java and the bridge node in Python uses the *ZeroMQ* protocol. The sensing action computed by the path planner is similarly published by the bridge node as a topic.

A ROS node that interfaces with the UAV control station was used to convert the sensing actions into UAV control commands. This node does not support a

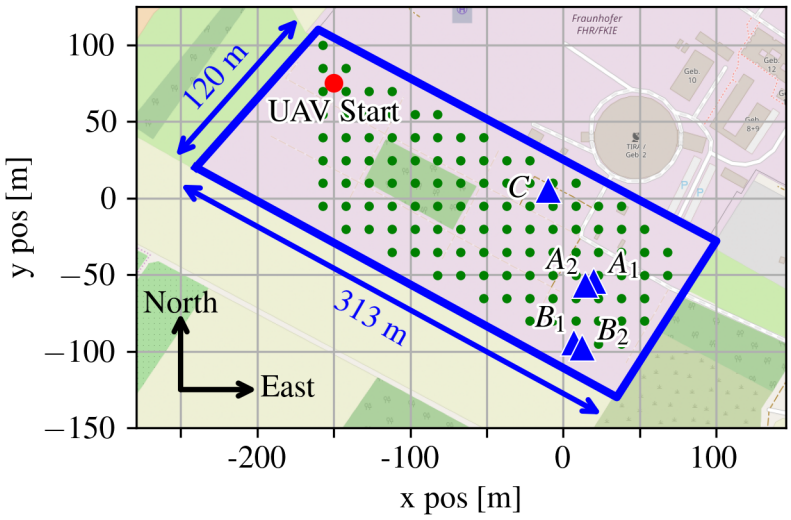



**Figure 6.9:** Experimental area. $A = \{A_1, A_2\}$, $B = \{B_1, B_2\}$, and $C$ denote different target positions, as measured by GPS. For different trial days the specific GPS position varied by a small bit and the differences are indicated by the index of $A$ and $B$. The ground station is located at the coordinate origin. Green dots indicate the possible sensing actions $\mathbb{A}$. The UAV start position corresponds to action $\mathbf{a}_0$. Map data © OpenStreetMap contributors.

dedicated full rotation of the UAV. We therefore divide the rotation into a sequence of three waypoints with different headings. This leads to a rotation based on three parts (see Figure 6.8). Before starting the rotation, the platform is kept stationary for one second to ensure it is in a stable position.

### 6.3.3 Experimental area

The experimental area is approximately 313 meters long and 120 meters wide. We discretized the area into possible measurement locations, which correspond to the action space $\mathbb{A}$. Figure 6.9 shows the experimental area and the action space. In total there are 112 sensing actions from which the planner can choose, with a distance of 15 meters each. The ground station was placed at the origin of a local east-north-up (ENU) coordinate system. At the same location the human pilot was present, to perform take-off and landing, as well as to interrupt the UAV in case of emergencies. Note that the sensing actions do not extend to the western border, as otherwise a tree line in the experimental area would have obstructed the line of sight between the human pilot and the UAV.

**Table 6.1:** Parameters of the rollout path planner

| Parameter | Value | Description |
|---|---|---|
| $\sigma$ | 7° | Measurement standard deviation |
| $\overline{\mu}$ | 10 m | Localization accuracy threshold |
| $\underline{r}$ | 50 m | Minimal distance to the confidence region |
| $v^s$ | 7 m/s | Platform speed |
| $t^m$ | 30 s | Duration of a measurement |
| $N$ | 1000 | Computational budget |
| $\underline{\zeta}$ | 0.7 | Cosine similarity threshold |

## 6.4 Experimental results

We performed six flights on three different days to analyse the system. Table 6.2 shows an overview over those flights. Additional flights were taken previously for debugging and setup of the sensor system. Flights 1-3 were made to determine suitable parameters for the path planner. Those flights were made partially manually controlled and partially controlled by the rollout path planner. Afterwards, the parameters of the path planner were fixed to those shown in Table 6.1. All flights were made at an altitude of approximately 20 m. Flights 1-3 also included additional debugging of the path planner. However, the direction finding algorithm was the same during flights 1-6. Therefore, the first three flights are included in the statistics about the direction finding performance.

For flights 4-6, target positions $B$ and $C$ were selected to increase the number of available sensing actions for the planner. As measurements need to be taken at least 50 m away from the confidence region, a centrally placed target prohibits more sensing actions than one on the border of the area. This is due to the restricted size of our experimental area.

### 6.4.1 Localization attempts

We refer to a segment of a flight which is controlled by the path planner and has the intent to localize the target as an *localization attempt*. In total, eleven localization attempts were made, which are shown in Table 6.3. All of those began with the UAV taking a measurement at the same location, shown in Figure 6.9.

**Table 6.2:** Flights

| Flight | Target position | Duration | Avg. Wind | Max Wind | #Msr |
|---|---|---|---|---|---|
| 1 | $A_1$ | 11:56 | - | - | 12 |
| 2 | $A_1$ | 20:14 | - | - | 13 |
| 3 | $A_2$ | 21:13 | 5.6 km/h | 14.3 km/h | 32 |
| 4 | $B_1$ | 19:05 | 6.5 km/h | 15.0 km/h | 19 |
| 5 | $C$ | 18:17 | 2.6 km/h | 5.6 km/h | 16 |
| 6 | $B_2$ | 14:23 | 3.2 km/h | - | 14 |

The flight durations are given in [mm:ss]. The average and maximal wind speed was measured by an anemometer at the ground station. No wind speed measurements were taken for flight 1 and 2. The anemometer showed an error message for the maximal wind speed on flight 6. Each group of two flights was conducted on a separate day. The column *# Msr* contains the total number of measurements made in this flight. Flight 5 also contains two measurements not belonging to a localization attempt, which therefore do not appear in Table 6.3.

**Table 6.3:** Localization attempts

| Loc. | Target position | Flight | T.u.l. | RMSE | Exp. RMSE | $\mathcal{C}^{95}$ | #Msr |
|---|---|---|---|---|---|---|---|
| 1 | $B_1$ | 4 | 03:18 | 24.34 m | 9.88 m | - | 4 (+1) |
| 2 | $B_1$ | 4 | 02:34 | 15.75 m | 9.10 m | ✓ | 4 |
| 3 | $B_1$ | 4 | 03:39 | 1.61 m | 9.80 m | ✓ | 4 (+1) |
| 4 | $B_1$ | 4 | 03:52 | 15.94 m | 8.31 m | - | 5 |
| 5 | $C$ | 5 | 00:23 | 168.27 m | 7.91 m | - | 1 |
| 6 | $C$ | 5 | 00:21 | 174.27 m | 8.97 m | - | 1 |
| 7 | $C$ | 5 | 02:56 | 7.76 m | 9.65 m | ✓ | 4 |
| 8 | $C$ | 5 | 02:52 | 10.47 m | 8.94 m | ✓ | 4 |
| 9 | $C$ | 5 | 03:18 | 7.98 m | 9.89 m | ✓ | 4 |
| 10 | $B_2$ | 6 | 04:44 | 9.46 m | 9.16 m | ✓ | 6 |
| 11 | $B_2$ | 6 | 05:12 | 9.10 m | 9.76 m | ✓ | 6 (+2) |

*Loc.* is the number of the localization attempt. *T.u.l.* is the time until localization, given in [mm:ss] from the start of the first measurement process to the announcement of the successful localization. The RMSE $\|\tilde{\mathbf{x}}_{\mathrm{K}}^t - \mathbf{x}^t\|_2$ is the distance between the point estimate and the true target position and the expected RMSE $\mu(\tilde{\mathbf{x}}_{\mathrm{K}}^t, b_{\mathrm{K}})$ is given by (3.58). The column $\mathcal{C}^{95}$ indicates whether the target was in the confidence region $\mathcal{C}^{95}(b_{\mathrm{K}})$. Finally, *#Msr* indicates the number of valid measurements passed to the path planner. In parentheses is the count of measurements that were classified as outliers. Localization attempts separated by a horizontal line correspond to the same flight.

In each localization attempt, the path planner terminated and a localization was returned with the expected RMSE below the localization accuracy threshold $\overline{\mu}$. In seven localization attempts the target was in the indicated confidence region. In two localization attempts (1 & 4) this was not the case, however, the target was still close to the point estimate. Two localization attempts (5 & 6) resulted in a completely wrong point estimate, which was due to the corresponding initial measurements being outliers. These outliers, together with the prior that the target is in the experimental area, reduced the uncertainty sufficiently that the target was considered to be localized after the first measurement. Section 4.3.1 describes an outlier detection criterion for the localizer, which detects a problem if the $4\sigma$ cones of the bearing measurements do not intersect. This criterion was not triggered here, because of the localization with only one measurement. These measurements also passed the outlier detection step described in this chapter. Outliers are further discussed in the next section.

A localization attempt typically required 4-6 measurements and took an average time of 3:53 minutes each to localize the target at position $B$ and 3:02 to localize the target at position C (excluding localization attempts 5 & 6). The paths taken by the UAV are shown in Figures 6.10 and 6.11 for target position $B$ and $C$, respectively. It can be seen that the path planner took measurements from multiple directions, trying to be as close as possible to the target while keeping the 50 m distance. Figure 6.10 also illustrates how the path planner adaptively reacted to the received measurements. Measurements taken in flight 4 (localization attempts 1-4) were all slightly more to the right, than measurements taken at flight 6 (localization attempts 10 & 11). Therefore, in flight 4 the path planner always flew to a measurement location at the north-eastern border of the experimental area, being convinced that this was sufficiently far away from the target. In comparison, in flight 6 the path planner always took a measurement location at the south-western border.

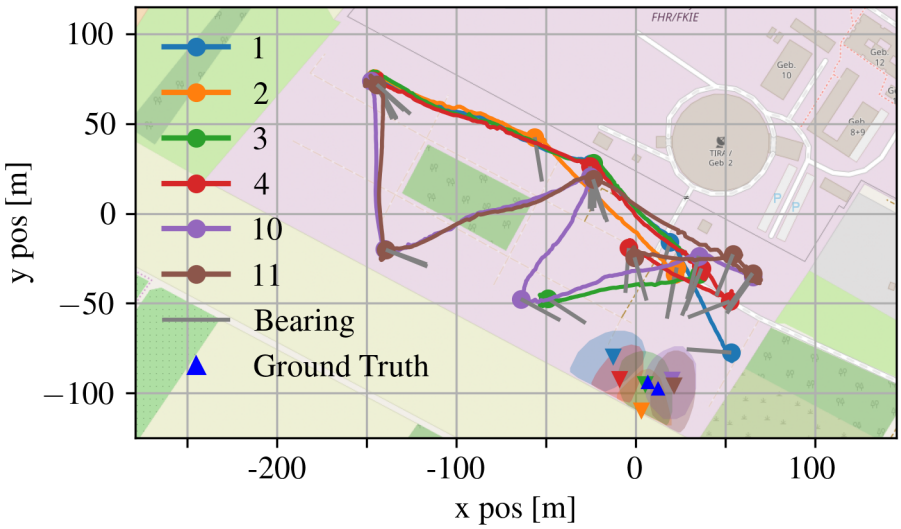



**Figure 6.10:** Flight paths to localize the target at position B. The true target positions are indicated by the upright blue triangles. The left one corresponds to $B_1$, the right one to $B_2$. Point estimates are indicated by the downward triangles. The shaded areas correspond to the confidence region $\mathcal{C}^{95}(b_{\mathrm{K}})$. Map data © OpenStreetMap contributors.

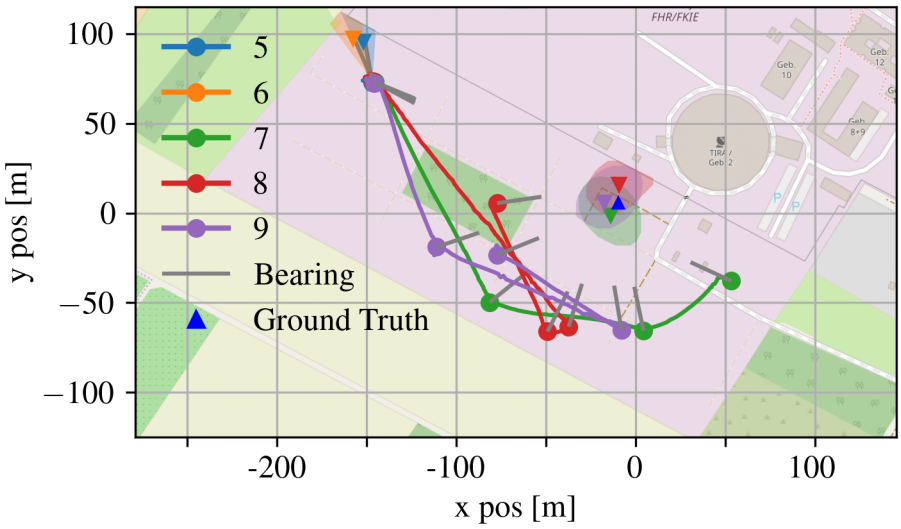



**Figure 6.11:** Flight paths to localize the target at position C. The true target position is indicated by an upright blue triangle. Point estimates are indicated by the downward triangles. The shaded areas correspond to the confidence region $\mathcal{C}^{95}(b_{\mathrm{K}})$. Map data © OpenStreetMap contributors.

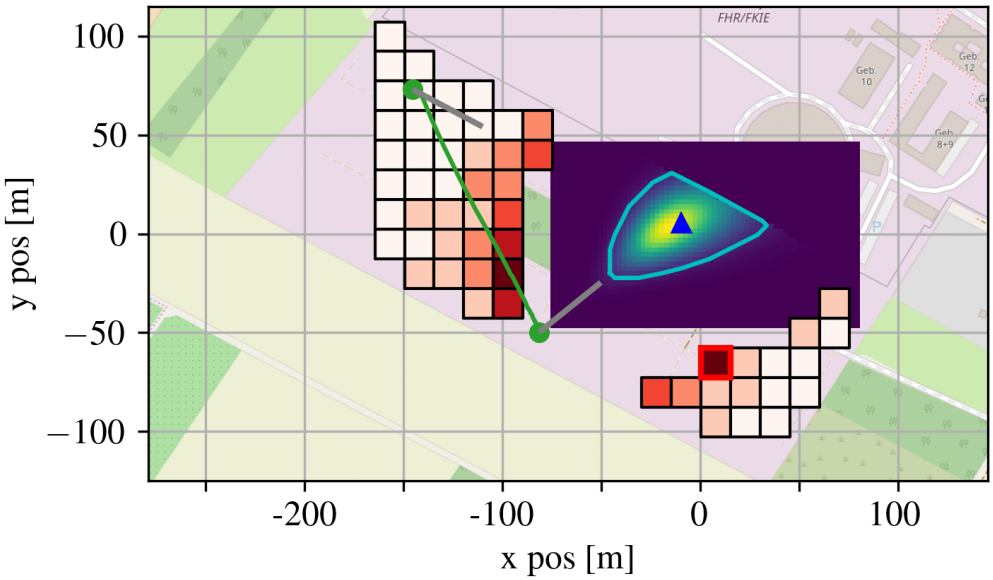



**Figure 6.12:** Action evaluation in localization attempt 7, after the second measurement. The colour of the action indicates in which of the six iterations it was eliminated. The selected action has a red border. The green line corresponds to the path of the UAV, gray are the measurements, the blue triangle is the true target position and cyan is $\mathcal{C}^{95}(b_2)$. Map data © OpenStreetMap contributors.

Figure 6.12 illustrates a single execution of the path planner. Only actions sufficiently far away from the confidence region were evaluated. The actions were evaluated in multiple rounds, as the sequential halving algorithm focused on the most promising actions. The colour of the actions indicates the iteration in which they were eliminated. As the sequential halving algorithm eliminates half of the remaining actions in each iteration, it took six iterations for 56 viable actions in this decision step. The adaptive action evaluation quickly focused on actions close to the confidence region. As the sequential halving algorithm uses the same number of samples in each round, the more promising actions were evaluated with a much greater number of rollouts than less promising actions. Actions eliminated in the first iteration were evaluated only with two rollouts, while the two last remaining actions were evaluated with in total 165 rollouts each.

### 6.4.2 Bearing measurements

Given a measurement $z$ and true target bearing $\theta$, the *measurement error* is defined as the difference $\theta - z \in [-\pi, \pi]$. A positive measurement error indicates that the measurement is to the right of the target bearing. The *absolute measurement error* is the absolute value of the measurement error.

In total, the data from all flights contains 106 measurements. Of those, 19 are outliers, defined as having an absolute measurement error greater than 30°. Of the 13 measurements taken at a distance to the target below 50 m, 10 measurements are outliers indicating the high likelihood of a false measurement at short distances. Figure 6.13 shows the measurement error with respect to the cosine similarity criterion for each measurement. Again, it can be seen that measurements below a distance of 50 m are mostly random, even if the cosine similarity does not predict a bad measurement. For measurements taken from farther away, the cosine similarity threshold of $\underline{\zeta} = 0.7$ rejects most of the outliers. Together, this justifies the approach of a cautious planner, which does both, try to keep a sufficient distance from the target and reject measurements that do not match the antenna pattern. There are two outliers from a farther distance that are not detected. These correspond to the initial measurements during localization attempts 5 and 6. Both point directly into the corner of the experimental area, which is surrounded by a metal fence. A likely explanation is that reflections from the fence caused multipath effects that misled the sensor.

Figure 6.14 shows the measurement errors of all measurements sufficiently far away ($> 50$ m), with a cosine similarity $> 0.7$ and without those two outliers. The average measurement error, or *bias*, is 0.17° and the standard deviation is 7.47°. This approximately matches the assumptions of the planner. However, the distribution varies between different flights, with especially flight 4 having larger measurement errors. This is reflected in the results of localization attempts 1 and 4, where the target is not in the confidence region $\mathcal{C}^{95}(b_K)$. It is unclear why the measurement quality differs from flight to flight. One possible explanation is a varying GPS quality, resulting in a worse GPS heading, and therefore less reliable direction output of

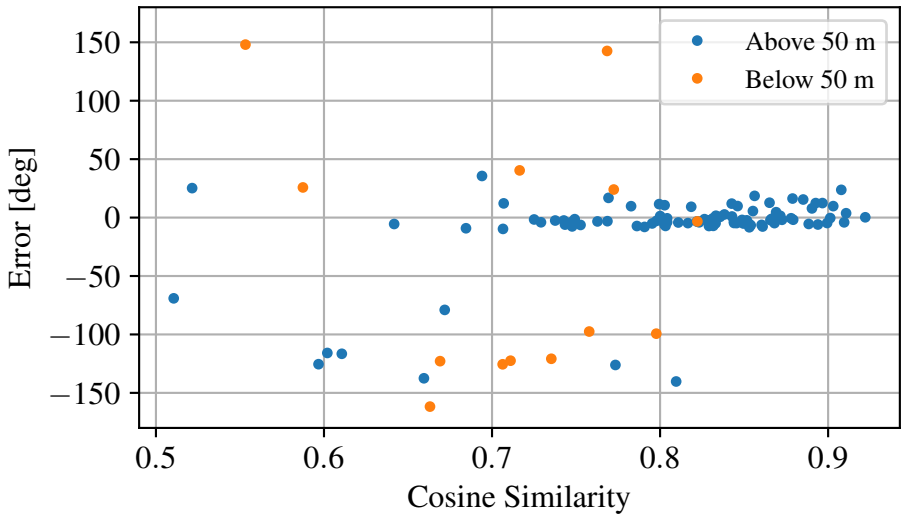



**Figure 6.13:** Measurement error dependent on the cosine similarity criterion.

the INS. The bearing estimator also explicitly tries to model the received pattern as a superposition of multiple signals and often returned multiple bearings. In this case, the sensor system selects the bearing with the highest weight and discards the others. A further tuning of the direction finder parameters [Krestel et al., 2019], might force the direction finder to include fewer bearings of potentially higher quality.

The assumptions of the planner on the measurements were determined after flights 1 and 2 and the cosine similarity threshold $\underline{\zeta}$ was determined after flight 3.

### 6.4.3 Time prediction accuracy

The cost function predicts the duration of different sensing actions. This is used by the path planner to select those sensing actions that lead to the minimal expected time until localization. It is therefore important that the cost function corresponds to the real duration of a sensing action. This section analyses how accurately the prediction of the cost function corresponds to the empirical durations encountered in the experiments.

Figure 6.15 shows the characteristics of a single movement of the UAV between two measurement locations. While the cost function assumes a linear movement of the UAV, the figure shows that a single movement consists of different phases. First, the UAV rotates in the direction of its travel. This behaviour is due

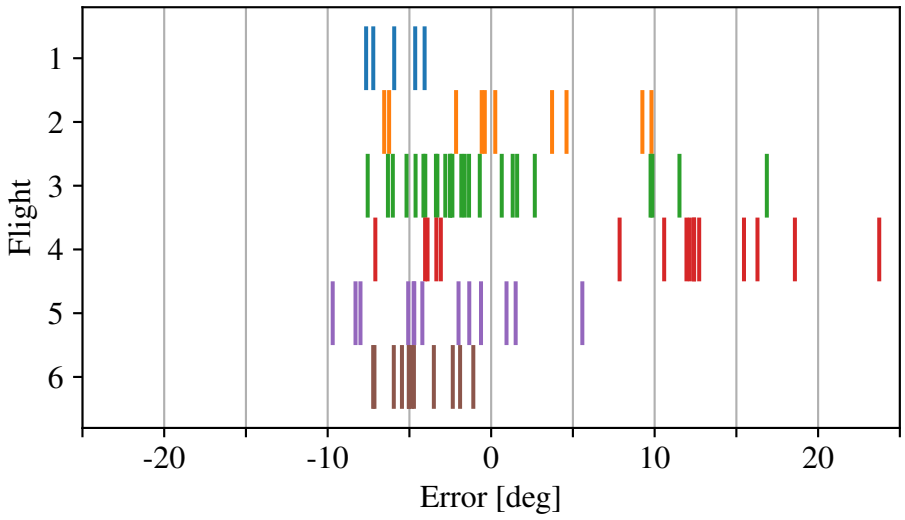



**Figure 6.14:** Measurement error for measurements with $> 50\,\text{m}$ distance and a cosine similarity $> 0.7$, excluding the two outliers. In total 81 measurements. The average measurement errors amount to $e_1 = -5.90°$, $e_2 = 1.17°$, $e_3 = -0.24°$, $e_4 = 8.30°$, $e_5 = -3.13°$, $e_6 = -4.52°$ and the standard deviations to $\sigma_1 = 1.56°$, $\sigma_2 = 5.68°$, $\sigma_3 = 6.09°$, $\sigma_4 = 9.48°$, $\sigma_5 = 4.36°$, $\sigma_6 = 1.96°$.

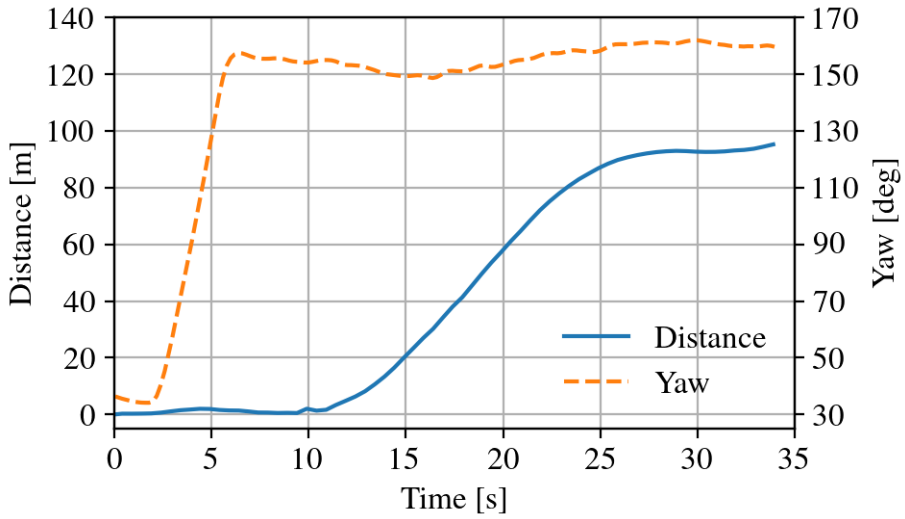



**Figure 6.15:** Orientation and distance traveled during a movement phase.

to a limitation of its API and is not requested by the path planner. Then it accelerates to its maximal speed, keeps this speed for some time, and decelerates before reaching the target position.

This indicates that the travel time is only roughly approximated by a simple linear model. As the cost function consists of a linear and a constant part, a question is whether the measurement duration $t^m$ can also be used to mitigate the inaccuracies of the travel time approximation. If the travel time can be modelled as

$$t^d(\mathbf{x}_k^s, \mathbf{a}_k) = \frac{\|\mathbf{x}_k^s - \mathbf{a}_k\|_2}{v^s} + t^{d,0} \tag{6.6}$$

where $t^{d,0}$ is a constant offset, a linear model would be feasible. A least square fit using the data from the experiments results in a constant term of $t^{d,0} = 11.2\,\mathrm{s}$. Figure 6.16 compares the predictions of a linear model with $7\,\mathrm{m/s}$ speed and a constant offset of $t^{d,0}$ with the measured travel times from the experiment. The RMSE of this linear model is 4.79 s. Based on the experimental data an improved model was created, which assumes that the UAV first rotates with 30°/s into its target orientation, then constantly accelerates with $1\,\mathrm{m/s^2}$ until either half of the way or its maximal speed $\bar{v}^s = 8.0\,\mathrm{m/s}$, decelerates before the target position and has a constant offset of 3 s. The RMSE of this model is 3.02 s. While this model is more accurate, the error of a linear model is not significantly higher. However, the linear approximation has a systematic error and consistently overestimates the travel costs of sensing actions with short travel times, due to the constant offset $t^{d,0}$.

The measurement duration is 21.4 s on average, with 2.9 s standard deviation. The termination criteria of the measurement rotation during flight 1 and 2 were different and in rare cases required manual intervention. This average is therefore computed using only flights 3 to 6, in which the measurements were fully automatic. An execution of the planner requires 0.56 s, with 0.35 s standard deviation.

After considering the movement time and measurement time in isolation, we now analyse the complete time of a sensing action. Figure 6.17 shows the prediction from the cost function (4.7) for sensing actions performed in the experiments. The cost function is parameterized using the values from Table 6.1. The prediction consists of movement and measurement time and is set in relation to the actual duration of the sensing actions. The RMSE of the prediction amounts to 14.1 s. An improved fit uses $t^m = 41.2\,\mathrm{s}$, which leads to an RMSE of 8.5 s. The duration of short sens-

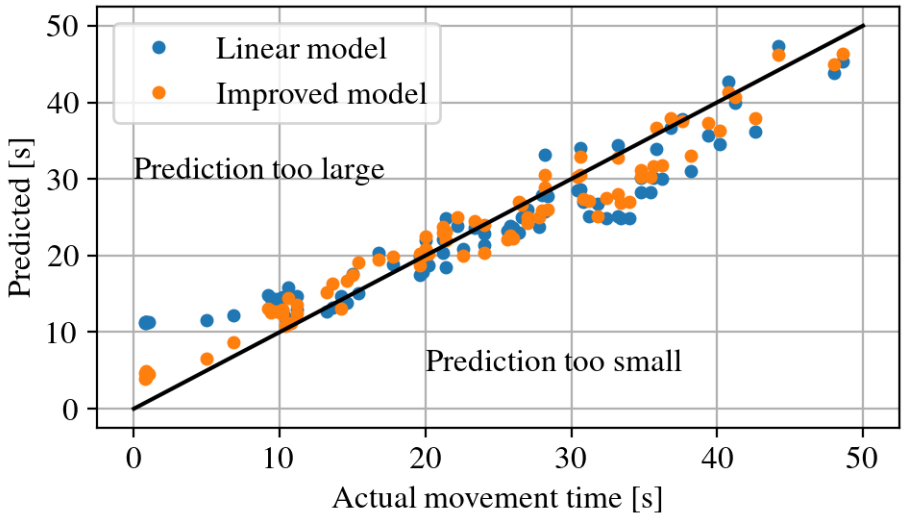



**Figure 6.16:** Evaluation of the accuracy of the movement time prediction. The linear model is a best-fit with $v^s = 7$ m/s. The improved model considers also acceleration and rotation.

ing actions is overestimated, which mirrors the behaviour seen in Figure 6.16 and indicates that a better movement model would improve the prediction of the cost function as well. There are some outliers (A-C) visible in the plot, which correspond to measurement steps with miss-detections and an additional measurement, making them approximately 21.4 s longer than expected. The fourth miss-detection (see Table 6.3) happened at the begin of localization attempt 11, and therefore its cost had never been predicted by the path planner.

Overall, a linear prediction model seems to be an acceptable approximation, but it overestimates the costs of shorter sensing actions which are those with less travel time. An improved model would potentially make those sensing actions more attractive to the path planner.

## 6.5 Conclusion

This chapter presented a sensor system for autonomously localizing an RF emitting target with a UAV. When adapting the algorithm from Chapter 4 and 5 to this system, we encountered three major challenges. First, it is required that the measurement location is farther than a minimal distance away from the true target position.

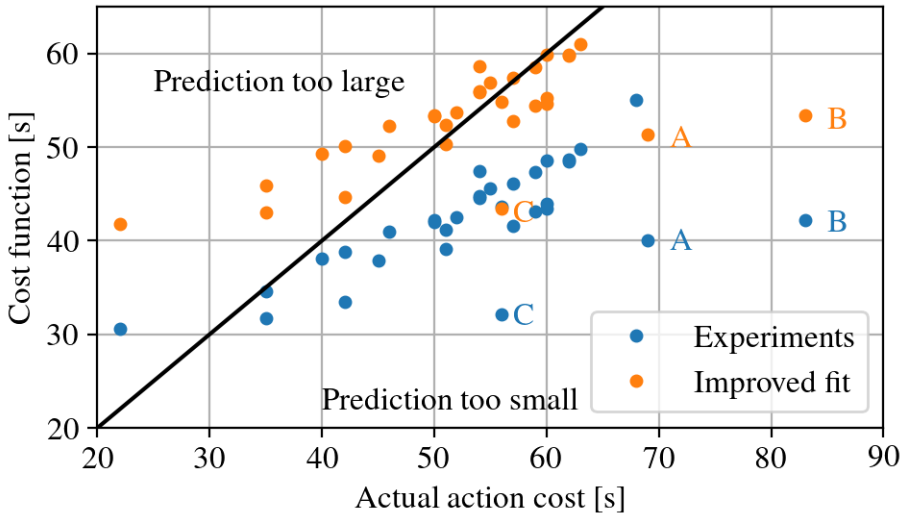



**Figure 6.17:** Comparison between the value of the cost function, and the actual duration of a sensing action. In the experiments a measurement duration of $t^m = 30\,\mathrm{s}$ was used. The improved fit uses a longer measurement duration of $t^m = 41.2\,\mathrm{s}$.

Second, the base policy of Chapter 4 and 5 is not feasible for a constrained action space. Third, occasionally outliers occur, which need to be detected and rejected.

The path planner was changed to solve the first problem by computing a confidence region that contains 95% of the probability mass of the belief. Then the action space $\mathbb{A}$ is limited to those actions $\mathbb{A}_k$ that are sufficiently distant from the confidence region. The second problem was solved by introducing a new base policy, which selects sensing actions greedily on the limited action space $\mathbb{A}_k$. When outliers occur, the received power over angle looks different than the antenna pattern. Therefore, the third problem was solved by rejecting measurements that do not match the antenna pattern. This decision is made based on the cosine similarity criterion.

We evaluated the rollout path planner and the sensor system in a total of six flights, and eleven localization attempts. From those eleven localization attempts, seven were fully successful, with the true target position in the indicated confidence region. Two flights led to a point estimate close to the true target position, but with the target being outside of the confidence region. Two flights led to a wrong point

estimate, due to significant outliers in the measurements. A probable explanation for this is a multipath reflection.

A comparison of the predicted duration of the sensing actions with the empirical duration from the experiments showed that the linear time prediction tends to overestimate the cost of short sensing actions. However, a more complex movement model can compensate for those differences.

In summary, this chapter made the following contributions:

1. It was demonstrated that the rollout path planner works in an experimental sensor system.

2. The experiments demonstrated a nonmyopic stochastic path planner. To the author's knowledge, this was the first demonstration of a nonmyopic, stochastic path planner that explicitly considered the specific trade-off between time spent travelling and time spent measuring.

# Chapter 7

# General Conclusions

In this thesis, the problem of emitter localization with a small unmanned aerial vehicle was considered. The goal was to implement effective sensor path planning to localize an emitter as fast as possible. A novel sensor path planning algorithm was created, and evaluated in simulations. In particular, the problem of action selection was considered. Finally, an experimental sensor system was built on which the path planner was tested.

Chapter 4 described a novel path planner, based on the policy rollout principle. This rollout path planner is stochastic, which means that it takes into account the uncertainty of the current belief. It evaluates different sensing actions by predicting the future development of the belief, based on different target positions and measurements. This prediction is performed until the target is localized, making it a nonmyopic path planner with a planning horizon spanning over the whole problem. The action selection is based on allocating a uniform number of evaluations to each action. The action evaluation is performed based on deterministic samples of the target position and an assumption that the measurement noise is zero in the future.

The localizer itself is also important. As the problem is highly nonlinear and in the beginning none, or only a single bearing measurement is available, a grid-based Bayes filter is used. The path planner needs to predict and update the belief for multiple future time steps, measurement realizations, and possible target positions. This leads to a high computational cost. An adaptive localizer, whose grid resolu-

tion is adaptively increased with improved location estimates, is created to reduce the computational cost.

The resulting rollout path planner was compared in simulations with path planners from the literature, where it showed on average a smaller time until localization.

In Chapter 5 the problem of action selection was analysed in detail. The analysis was separated into two subproblems: the evaluation of the expected value from one action and the search for the best action.

Three different methods were compared for the evaluation of the expected value: deterministic samples, as previously used in Chapter 4, plain Monte Carlo, and common random numbers. It was shown that deterministic samples work well, but the performance gain with more samples is not always monotonic. Common random numbers and plain Monte Carlo showed a monotonically increasing performance with more samples leading to better results.

Five methods to search for the best action were evaluated: splitting the computational budget uniformly over the actions, stochastic gradient descent, BFGS, quadrant search, and sequential halving. For a small computational budget quadrant search showed good results, and sequential halving for a higher computational budget. The results of Monte-Carlo based action selection methods were well-behaved: for a higher computational budget, the performance increased. However, deterministic samples did not show a monotonic improvement. In a sensitivity analysis the results replicated for different parameters, and also replicated on a different scenario with a different initial belief and a slightly different localizer. An analysis on the correlation between choosing an action with a better action value and the time until the target is localized showed a clear correlation.

The results of an experimental evaluation were presented in Chapter 6. An experimental setup, consisting of a UAV, its payload, a ground station, and an emitter, was created to evaluate the rollout path planner. Based on the results of the previous chapter, sequential halving and common random numbers were used for the experiments. The experimental sensor system showed additional limitations, which needed to be modelled: no measurements are feasible at close range and occasion-

ally outliers happen. The first problem was taken into account by restricting the action space to those actions sufficiently far away from the uncertainty region. This way it is unlikely that the target is close to the UAV when taking a measurement. The second problem was taken into account by an additional outlier detection step, based on the cosine similarity measure. Measurements that were not sufficiently similar to the antenna pattern were rejected.

Six flights to evaluate the sensor system and eleven localization attempts were made in total. From these, seven were fully successful, in the sense that the emitter was in the indicated uncertainty region and two were successful in the sense that the indicated localization was close to the true target position. Outliers that were not detected by the cosine similarity criterion happened in two cases. Due to the geometry of the experimental area, a likely explanation is that those were due to multipath reflections. The measurements were analysed and an average standard deviation of $7.47^\circ$ was found for the measurement noise. However, the distribution varied between flights. In addition, the model that predicted the time cost of a single sensing action was analysed. It was seen that a linear model as used in the experiments showed acceptable performance, but systematically underestimated some actions. An improved model was created, which takes the rotation and acceleration of the UAV into account, and could predict the time of a sensing action slightly better.

## 7.1 Contributions

This thesis provides the following contributions to the literature.

**Creation of a novel path planner for localizing an emitter with a UAV.** A novel path planner for localizing an emitter with a UAV was created. The algorithm takes into account the special measurement characteristics of the sensor system, which consists of the requirement of staying stationary during the measurement process. Compared to similar path planners from the literature, the rollout path planner takes the uncertainty of the state estimate and future measurements explicitly into account. This results in a better performance compared to those algorithms, as shown in simulations. The rollout path

planner also uses a novel adaptive grid-based localizer, which uses a convex hull to focus the grid on possible target locations.

**Optimization of the action selection in a policy rollout algorithm.** Previous uses of the policy rollout algorithm do not consider in detail the step of action selection. Only a minimal subset of the applications use any special optimizer at all. In this thesis, a connection between the policy rollout action selection and the problem of minimizing stochastic functions was drawn. Several action selection methods were evaluated, some of which have never been used in the policy rollout context before. In particular, to the author's knowledge, this work is the first to consider continuous action space for a policy rollout method. The trade-off between finding the best action and computational budget was analysed for the developed path planner and as result an informed decision can be made about the required computational budget.

**Analysis of the relationship between action selection and total received cost.** It is theoretically shown that because of the rollout improvement property, the rollout policy receives lesser cost than the base policy if the action with minimal action value is chosen. However, with Monte Carlo evaluation, it cannot be guaranteed that the true best action is chosen. To the author's knowledge, this work is the first to analyse the performance when, due to a limited computational budget, a non-optimal action is selected in the policy rollout algorithm. In this thesis a correlation between finding the minimum of the predicted action value and the total received cost was found. Therefore, selecting an action with a better action value in the developed path planner leads to a reduced time until the target is localized, even if the selected action is not optimal. This leads to the conclusion that the use of effective search methods for action selection influences the overall rollout performance and should thus be further studied.

**Demonstration of the novel path planner in an experimental sensor system.** An experimental sensor system with a UAV and a single directional antenna

was developed for testing the rollout path planner developed in this thesis. The path planner was also modified for real-world use, based on additional limitations such as minimal measurement distance and occasional outliers identified during the test flights. The experimental results demonstrate that a modified version of the rollout path planner can be used to localize emitters autonomously. Based on the experimental results, improved time prediction models were developed.

## 7.2 Future work

As listed in the previous section, this thesis provides original contributions to the research field of sensor path planning. However, the work also opened up new research questions which can be considered in the future.

One direction in which the work of this thesis can be extended is by looking at different and more challenging scenarios. One possible extension would be to consider the localization of multiple emitters. First steps in this direction have been taken in a supervised masters thesis [Krestel, 2021], but an experimental validation is still missing. Another extension would be the use of multiple UAVs, in which case the cooperation between the UAVs need to be considered. Using multiple UAVs would likely lead to faster localizations. While there exists a significant amount of work for optimizing sensor systems of multiple UAVs with continuous measurements, a sensor system as the one described in this thesis would lead to measurements at different times. Likely one UAV would be flying, while the other takes a measurement. The path planner needs to decide on the sensing action of one UAV, before the results of the other UAV's sensing action is received. Planning under these circumstances would be a challenging problem.

As the experiments have shown, sometimes outliers cannot be detected due to their signal properties alone. An interesting area of future research would be to make the planner more resistant against such outliers. A way to achieve this could be to require a minimal number of measurements from different positions for a target localization. If those measurements are inconsistent, the localizer could create

different hypotheses about which measurements are outliers and which are valid. Then the path planner could select the next sensing actions in a way that allows to separate the correct and the wrong hypotheses. There is always a mismatch between model and reality and likely it is not possible to fully remove the possibility of undetected outliers. But it can be expected that such a system reduces the influence of single outliers.

In this thesis the evaluation of action selection methods was focused on the problem of sensor path planning. However, this problem is relevant for every use of the policy rollout method when Monte Carlo evaluation is used to estimate the action values. The results of this thesis and the determined relative performance of the action selection algorithms are based on the path planning problem. A detailed investigation about the influence of different action selection methods in policy rollout algorithms for different application areas would be interesting. One step in this direction was made in another supervised master thesis [Gerlach, 2020], where aside from evaluating a Bayesian action selection algorithm on this application, the optimization of action selection in the *quiz problem* [Bertsekas and Castañón, 1999] was analysed with promising results.

# Bibliography


Joel A. E. Andersson, Joris Gillis, Greg Horn, James B. Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

Edson Hiroshi Aoki, Arunabha Bagchi, Pranab Mandal, and Yvo Boers. A theoretical look at information-driven sensor management criteria. In *Fusion 2011 - 14th International Conference on Information Fusion*, Chicago, IL, USA, 2011a. IEEE.

Edson Hiroshi Aoki, Arunabha Bagchi, Pranab Mandal, and Yvo Boers. On the "near-universal proxy" argument for theoretical justification of information-driven sensor management. In *2011 IEEE Statistical Signal Processing Workshop (SSP)*, pages 245–248. IEEE, June 2011b.

Sabine Apfeld, Alexander Charlish, and Wolfgang Koch. An adaptive receiver search strategy for electronic support. In *Sensor Signal Processing for Defence (SSPD)*, Edinburgh, UK, 2016. IEEE.

Jean-Yves Audibert and Sébastien Bubeck. Best arm identification in multi-armed bandits. In *COLT - 23th Conference on Learning Theory*, Haifa, Israel, 2010.

Peter Auer, Niolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

Yaakov Bar-Shalom, X.-Rong Li, and Thiagalingam Kirubarajan. *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, Inc., 2001.

Bradley M. Bell and Frederick W. Cathey. The iterated Kalman filter update as a Gauss-Newton method. *IEEE Transactions on Automatic Control*, 38(2):294–297, 1993.

Luca Bertazzi, Adamo Bosco, Francesca Guerriero, and Demetrio Laganà. A stochastic inventory routing problem with stock-out. *Transportation Research Part C: Emerging Technologies*, 27:89–107, 2013.

Dimitri P. Bertsekas. Differential training of rollout policies. In *Proceedings of 35th Allerton Conference on Communication, Control and Computing*, pages 1–10, Allerton Park, Illinois, USA, 1997.

Dimitri P. Bertsekas. Dynamic programming and suboptimal control: A survey from ADP to MPC*. *European Journal of Control*, 11(4-5):310–334, 2005.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II Approximate Dynamic Programming*. Athena Scientific, Belmont, Massachusetts, USA, 4th edition, 2012.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, Massachusetts, USA, 4th edition, 2017.

Dimitri P. Bertsekas and David A. Castañón. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5(1):89–108, 1999.

Dimitri P. Bertsekas, John N. Tsitsiklis, and Cynara Wu. Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3(3):245–262, 1997.

Dimitris Bertsimas and Ioana Popescu. Revenue management in a dynamic network environment. *Transportation Science*, 37(3):257–277, 2003.

Steffen Beyme and Cyril Leung. Rollout algorithms for wireless sensor network-assisted target search. *IEEE Sensors Journal*, 15(7):3835–3845, 2015.

Adrian N. Bishop, Bariş Fidan, Brian D.O. Anderson, Kutluyil Doğançay, and Pubudu N. Pathirana. Optimality analysis of sensor-target geometries in passive lo-

calization: Part 1 - bearing-only localization. *Proceedings of the 2007 International Conference on Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP*, 1:7–12, 2007.

Adrian N. Bishop, Barış Fidan, Brian D.O. Anderson, Kutluyıl Doğançay, and Pubudu N. Pathirana. Optimality analysis of sensor-target localization geometries. *Automatica*, 46(3):479–492, March 2010.

André Brandenburger, Folker Hoffmann, and Alexander Charlish. Co-training an observer and an evading target. In *24th International Conference on Information Fusion (FUSION)*, Rustenburg, South Africa, 2021.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–49, 2012.

Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5809 LNAI:23–37, 2009.

Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. X-armed bandits. *Journal of Machine Learning Research*, 12:1655–1695, 2011.

Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, September 1995.

Chenghui Cai and Silvia Ferrari. Information-driven sensor path planning by approximate cell decomposition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(3):672–689, 2009.

Eduardo F. Camacho and Carlos Bordons. *Model Predictive Control*. Springer, second edition, 2007.

Alexander Charlish and Folker Hoffmann. Anticipation in cognitive radar using stochastic control. In *2015 IEEE Radar Conference (RadarCon)*, pages 1692–1697, Arlington, Virginia, USA., 2015. IEEE.

Alexander Charlish and Folker Hoffmann. Cognitive radar management. In *Novel Radar Techniques and Applications Volume 2: Waveform Diversity and Cognitive Radar, and Target Tracking and Data Fusion*, chapter 5, pages 157–193. Institution of Engineering and Technology, 1 edition, 2017.

Alexander Charlish, Folker Hoffmann, Christoph Degen, and Isabel Schlangen. The development from adaptive to cognitive radar resource management. *IEEE Aerospace and Electronic Systems Magazine*, 35(6):8–19, June 2020.

Benjamin Charrow, Gregory Kahn, Sachin Patil, Sikang Liu, Ken Goldberg, Pieter Abbeel, Nathan Michael, and Vijay Kumar. Information-theoretic planning with trajectory optimization for dense 3D mapping. *Robotics: Science and Systems*, 11, 2015.

Chun-Hung Chen and Loo Hay Lee. *Stochastic Simulation Optimization - An Optimal Computing Budget Allocation*. World Scientific Publishing, Singapore, 2011.

Edwin K. P. Chong, Christopher M. Kreucher, and Alfred O. Hero III. Monte-carlo-based partially observable markov decision process approximations for adaptive sensing. In *Proceedings of the 9th International Workshop on Discrete Event Systems*, pages 173–180, Göteborg, Sweden, 2008. IEEE.

Edwin K. P. Chong, Christopher M. Kreucher, and Alfred O. Hero. Partially observable Markov decision process approximations for adaptive sensing. *Discrete Event Dynamic Systems*, 19(3):377–422, 2009.

I. Vaughan L. Clarkson. Optimisation of periodic search strategies for electronic support. *IEEE Transactions on Aerospace and Electronic Systems*, 47(3):1770–1784, 2011.

Oliver Cliff, Robert Fitch, Salah Sukkarieh, Debbie Saunders, and Robert Heinsohn. Online localization of radio-tagged wildlife with an autonomous aerial robot system. In *Robotics: Science and Systems XI*, Rome, Italy, 2015. Robotics: Science and Systems Foundation.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., Hoboken, NJ, USA, second edition, 2006.

John E. Dennis, Jr. and Jorge J. Moré. Quasi-Newton methods, motivation and theory. *SIAM Review*, 19(1):46–89, January 1977.

Kutluyil Doğançay. UAV path planning for passive emitter localization. *IEEE Transactions on Aerospace and Electronic Systems*, 48(2):1150–1166, 2012.

Louis Dressel and Mykel J. Kochenderfer. Pseudo-bearing measurements for improved localization of radio sources with multirotor UAVs. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6560–6565, Brisbane, Australia, 2018. IEEE.

Selim Engin and Volkan Isler. Active localization of multiple targets from noisy relative measurements. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 398–413, Oulu, Finland, 2020.

Dario Floreano and Robert J. Wood. Science, technology and the future of small autonomous drones. *Nature*, 521(7553):460–466, 2015.

Eli Fogel and Motti Gavish. Nth-order dynamics target observability from angle measurements. *IEEE Transactions on Aerospace and Electronic Systems*, 24(3): 305–308, 1988.

Mark R. Fuller and Todd K. Fuller. Radio-telemetry equipment and applications for carnivores. In Luigi Boitani and Roger A. Powell, editors, *Carnivore Ecology and*

*Conservation - A handbook of techniques*, chapter 7. Oxford University Press, 2012.

Victor Gabillon and Alessandro Lazaric. Rollout allocation strategies for classification-based policy iteration. In *ICML 2010 Workshop on Reinforcement Learning and Search in Very Large Spaces*, pages 0–3, Haifa, Israel, 2010.

Victor Gabillon, Mohammad Ghavamzadeh, and Alessandro Lazaric. Best arm identification: A unified approach to fixed budget and fixed confidence. In *Advances in Neural Information Processing Systems 25*, pages 3212–3220. Curran Associates, Inc., Lake Tahoe, Nevada, USA, 2012.

Aurélien Garivier and Emilie Kaufmann. Optimal best arm identification with fixed confidence. In *COLT - 29th Conference on Learning Theory*, pages 1–30, New York, USA, 2016.

Thore Gerlach. *Knowledge Gradient for Policy Rollout Algorithms*. Master thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2020.

Thore Gerlach, Folker Hoffmann, and Alexander Charlish. Policy rollout action selection with knowledge gradient for sensor path planning. In *24th International Conference on Information Fusion (FUSION)*, Rustenburg, South Africa, 2021.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Justin C. Goodson, Jeffrey W. Ohlmann, and Barrett W. Thomas. Rollout policies for dynamic solutions to the multivehicle routing problem with stochastic demand and duration limits. *Operations Research*, 61(1):138–154, February 2013.

Neil J. Gordon, David J. Salmond, and Adrian F. M. Smith. Novel approach to nonlinear/non-gaussian Bayesian state estimation. *Radar and Signal Processing, IEE Proceedings F*, 140(2):107–113, 1993.

Juergen Graefenstein, Amos Albert, Peter Biber, and Andreas Schilling. Wireless node localization based on RSSI using a rotating antenna on a mobile robot. In

*Proceedings of the 6th Workshop on Positioning, Navigation and Communication (WPNC)*, pages 253–259, Hannover, Germany, 2009. IEEE.

Maria S. Greco, Fulvio Gini, Pietro Stinco, and Kristine Bell. Cognitive radars: On the road to reality: Progress thus far and possibilities for the future. *IEEE Signal Processing Magazine*, 35(4):112–125, 2018.

Ben Grocholsky, Alexei Makarenko, and Hugh Durrant-Whyte. Information-theoretic coordinated control of multiple sensor platforms. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1521–1526, Taipei, Taiwan, 2003. IEEE.

Sevgi Zubeyde Gurbuz, Hugh D. Griffiths, Alexander Charlish, Muralidhar Rangaswamy, Maria Sabrina Greco, and Kristine Bell. An overview of cognitive radar: Past, present, and future. *IEEE Aerospace and Electronic Systems Magazine*, 34(12):6–18, December 2019.

Sherry E. Hammel, Pan-Tai Liu, Edward J. Hilliard, and Kai F. Gong. Optimal observer motion for localization with bearing measurements. *Computers & Mathematics with Applications*, 18(1-3):171–180, 1989.

Simon Haykin. Cognitive radar: A way of the future. *IEEE Signal Processing Magazine*, 23(1):30–40, 2006.

Ying He and Edwin K. P. Chong. Sensor scheduling for target tracking in sensor networks. In *43rd IEEE Conference on Decision and Control*, pages 743–748, Atlantis, Paradise Island, Bahamas, 2004. IEEE.

Ying He and Edwin K. P. Chong. Sensor scheduling for target tracking: A Monte Carlo sampling approach. *Digital Signal Processing*, 16(5):533–545, 2006.

Marcel L. Hernandez. Optimal sensor trajectories in bearings-only tracking. In *The 7th International Conference on Information Fusion (FUSION)*, pages 1–8, Stockholm, Sweden, 2004. IEEE.

Onésimo Hernández-Lerma and Jean Bernard Lasserre. *Discrete-Time Markov Control Processes: Basic Optimality Criteria*. Springer, 1996.

Alfred O. Hero III and Douglas Cochran. Sensor management: Past, present, and future. *IEEE Sensors Journal*, 11(12):3064–3075, 2011.

Folker Hoffmann and Alexander Charlish. A resource allocation model for the radar search function. In *International Radar Conference*, pages 1–6, Lille, France, 2014. IEEE.

Folker Hoffmann, Alexander Charlish, and Wolfgang Koch. Trajectory optimization for multi-platform bearing-only tracking with ghosts. In *Proceedings of the 19th International Conference on Information Fusion (FUSION)*, pages 39 – 44, Heidelberg, Germany, 2016a.

Folker Hoffmann, Matthew Ritchie, Francesco Fioranelli, Alexander Charlish, and Hugh Griffiths. Micro-doppler based detection and tracking of UAVs with multistatic radar. In *IEEE Radar Conference (RadarConf)*, pages 893–898, Philadelphia, PA, USA, 2016b. IEEE.

Folker Hoffmann, Hans Schily, Alexander Charlish, Matthew Ritchie, and Hugh Griffiths. A rollout based path planner for emitter localization. In *Proceedings of the 22nd International Conference on Information Fusion (FUSION)*, Ottawa, ON, Canada, 2019.

Folker Hoffmann, Alexander Charlish, Matthew Ritchie, and Hugh Griffiths. Sensor path planning using reinforcement learning. In *Proceedings of the 23nd International Conference on Information Fusion (FUSION)*, Rustenburg, South Africa (Virtual), 2020.

Folker Hoffmann, Alexander Charlish, Matthew Ritchie, and Hugh Griffiths. Policy rollout action selection in continuous domains for sensor path planning. *IEEE Transactions on Aerospace and Electronic Systems*, pages 2247–2264, 2021.

Gabriel M. Hoffmann and Claire J. Tomlin. Mobile sensor network control using mutual information methods and particle filters. *IEEE Transactions on Automatic Control*, 55(1):32–47, January 2010.

Colin Horne, Matthew Ritchie, Hugh Griffiths, Folker Hoffmann, and Alexander Charlish. Experimental validation of cognitive radar anticipation using stochastic control. In *Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, USA, 2016.

Nathan T. Hui, Eric K. Lo, Jen B. Moss, Glenn P. Gerber, Mark E. Welch, Ryan Kastner, and Curt Schurgers. A more precise way to localize animals using drones. *Journal of Field Robotics*, 38(July 2020):1–12, 2021.

Jason T. Isaacs, François Quitin, Luis R. García Carrillo, Upamanyu Madhow, and João P. Hespanha. Quadrotor control for RF source localization and tracking. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 244–252, Orlando, FL, USA, 2014. IEEE.

David Jun and Douglas L Jones. The value of sleeping: A rollout algorithm for sensor scheduling in HMMs. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 181–184, Austin, Texas, USA, December 2013. IEEE.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101 (1-2):99–134, 1998.

Rudolf E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35, 1960.

Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, June 2011.

Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, volume 28, pages 1238–1246, Atlanta, Georgia, USA, 2013.

Fotios Katsilieris, Yvo Boers, and Hans Driessen. Sensor management for PRF selection in the track-before-detect context. *IEEE National Radar Conference - Proceedings*, pages 0360–0365, 2012.

David J. Kershaw and Robin J. Evans. Optimal waveform selection for tracking systems. *IEEE Transactions on Information Theory*, 40(5):1536–1550, 1994.

Vesa Klumpp and Uwe D. Hanebeck. Dirac mixture trees for fast suboptimal multidimensional density approximation. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 593–600, Seoul, Korea, 2008. IEEE.

Wolfgang Koch. *Tracking and Sensor Data Fusion*. Mathematical Engineering. Springer, Heidelberg, Germany, 2014.

Levente Kocsis and Csaba Szepesvari. Bandit based Monte-Carlo planning. In *ECML'06 Proceedings of the 17th European conference on Machine Learning*, pages 282–293, 2006.

Thomas Kollar and Nicholas Roy. Trajectory optimization using reinforcement learning for map exploration. *The International Journal of Robotics Research*, 27(2):175–196, February 2008.

Fabian Körner, Raphael Speck, Ali Haydar Göktoğan, and Salah Sukkarieh. Autonomous airborne wildlife tracking using radio signal strength. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 107–112, Taipei, Taiwan, October 2010. IEEE.

Lucas W. Krakow, Edwin K. P. Chong, Kenneth N. Groom, John Harrington, Yun Li, and Brian Rigdon. Control of perimeter surveillance wireless sensor networks

via partially observable Marcov decision process. In *Proceedings of the 40th Annual International Carnahan Conference on Security Technology*, pages 1–8, Lexington, KY, USA, 2006.

Markus Krestel. *Implementierung und Analyse eines Spinning Direction-Finders*. Bachelor thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2018. Translated title: implementation and analysis of a spinning direction finder.

Markus Krestel. *Sensor Path Planning for Multi-Emitter Localization using Policy Rollout*. Master thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2021.

Markus Krestel, Folker Hoffmann, Hans Schily, Alexander Charlish, and Sven Rau. Passive emitter direction finding using a single antenna and compressed sensing. In *2019 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, pages 1–5, Bonn, Germany, 2019. IEEE.

Chris Kreucher, Alfred O. Hero, and Keith Kastella. A comparison of task driven and information driven sensor management for target tracking. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 4004–4009, Seville, Spain, 2005.

Thomas R. Kronhamn. Bearings-only target motion analysis based on a multihypothesis Kalman filter and adaptive ownship motion control. *IEE Proceedings - Radar, Sonar and Navigation*, 145(4):247, 1998.

Vija Y. Kumar and Nathan Michael. Opportunities and challenges with autonomous micro aerial vehicles. *International Journal of Robotics Research*, 31(11):1279–1291, 2012.

Serge Lang. *Linear Algebra*. Springer, third edition, 2004.

Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

Jean-Pierre Le Cadre and S. Laurent-Michel. Optimizing the receiver maneuvers for bearings-only tracking. *Automatica*, 35(4):591–606, April 1999.

Cindy Leung, Shoudong Huang, Ngai Kwok, and Gamini Dissanayake. Planning under uncertainty using model predictive control for information gathering. *Robotics and Autonomous Systems*, 54(11):898–910, November 2006.

Yun Li, Lucas W. Krakow, Edwin K. P. Chong, and Kenneth N. Groom. Approximate stochastic dynamic programming for sensor scheduling to track multiple targets. *Digital Signal Processing*, 19(6):978–989, December 2009.

Chris Mansley, Ari Weinstein, and Michael L. Littman. Sample-based planning for continuous action Markov decision processes. In *ICAPS 2011 - Proceedings of the 21st International Conference on Automated Planning and Scheduling*, pages 335–338, Freiburg, Germany, 2011.

David McClung and Peter Schaerer. *The avalanche handbook*. The Mountaineers Books, third edition, 2006.

Amy McGovern and Eliot Moss. Scheduling straight-line code using reinforcement learning and rollouts. In *Advances in Neural Information Processing Systems*, pages 903–909, Denver, CO, USA, 1999.

Scott A. Miller, Zachary A. Harris, and Edwin K. P. Chong. A POMDP framework for coordinated guidance of autonomous UAVs for multitarget tracking. *EURASIP Journal on Advances in Signal Processing*, 2009:724597, 2009.

Darko Mušicki. Bearings only single-sensor target tracking using gaussian mixtures. *Automatica*, 45(9):2088–2092, September 2009.

Steven Nardone and Vincent Aidala. Observability criteria for bearings-only target motion analysis. *IEEE Transactions on Aerospace and Electronic Systems*, AES-17(2):162–166, 1981.

Hoa Van Nguyen, Michael Chesser, Lian Pin Koh, S. Hamid Rezatofighi, and Damith C. Ranasinghe. TrackerBots: Autonomous unmanned aerial vehicle for real-time localization and tracking of multiple radio-tagged animals. *Journal of Field Robotics*, 36(3):617–635, 2019.

Hoa Van Nguyen, Fei Chen, Joshua Chesser, Hamid Rezatofighi, and Damith Ranasinghe. LAVAPilot : Lightweight UAV trajectory planner with situational awareness for embedded autonomy to track and locate radio-tags. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2488–2495, Las Vegas, NV, USA, 2020.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer New York, 2006.

Clara Novoa and Robert Storer. An approximate dynamic programming approach for the vehicle routing problem with stochastic demands. *European Journal of Operational Research*, 196(2):509–515, 2009.

Marc Oispuu. *Passive Emitter Localization by Direct Position Determination with Moving Array Sensors*. Phd, Universität Siegen, 2013.

Yaakov Oshman and Pavel Davidson. Optimization of observer trajectories for bearings-only target localization. *IEEE Transactions on Aerospace and Electronic Systems*, 35(3):892–902, 1999.

Jean-Michel Passerieux and Dominique Van Cappel. Optimal observer maneuver for bearings-only tacking. *IEEE Transactions on Aerospace and Electronic Systems*, 34(3):777–788, 1998.

Liam Paull, Sajad Saeedi, Mae Seto, and Howard Li. Sensor-driven online coverage planning for autonomous underwater vehicles. *IEEE/ASME Transactions on Mechatronics*, 18(6):1827–1838, 2013.

Warren B. Powell. *Approximate Dynamic Programming*. John Wiley & Sons, Inc., Hoboken, New Jersey, second edition, 2011.

Warren B. Powell. A unified framework for stochastic optimization. *European Journal of Operational Research*, 275(3):795–821, 2019.

Luc Pronzato and Andrej Pázman. *Design of Experiments in Nonlinear Models*. Springer, 2013.

Martin L. Puterman. *Markov Decision Processes*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2005.

Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source robot operating system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

Edward Raff. JSAT: Java statistical analysis tool, a library for machine learning. *Journal of Machine Learning Research*, 18(23):1–5, 2017.

Shankarachary Ragi and Edwin K. P. Chong. UAV path planning in a dynamic environment via partially observable Markov decision process. *IEEE Transactions on Aerospace and Electronic Systems*, 49(4):2397–2412, 2013.

Shankarachary Ragi, Hans D. Mittelmann, and Edwin K. P. Chong. Directional sensor control: Heuristic approaches. *IEEE Sensors Journal*, 15(1):374–381, January 2015.

Branko Ristic, Mark Morelande, and Ajith Gunatilaka. Information driven search for point sources of gamma radiation. *Signal Processing*, 90(4):1225–1239, April 2010.

Reuven Y. Rubinstein and Dirk P. Kroese. *Simulation and the Monte Carlo Method*. Wiley, third edition, 2016.

Allison D. Ryan, Hugh Durrant-Whyte, and J. Karl Hedrick. Information-theoretic sensor motion control for distributed estimation. In *Proceedings of ASME 2007 International Mechanical Engineering Congress and Exposition (IMECE)*, pages 725–734, Seattle, Washington, USA, 2007. ASME.

Anshu Saksena and I-Jeng Wang. Dynamic ping optimization for surveillance in multistatic sonar buoy networks with energy constraints. In *47th IEEE Conference on Decision and Control*, pages 1109–1114, Cancun, Mexico, 2008. IEEE.

Peter Sarunic and Rob Evans. Hierarchical model predictive control of UAVs performing multitarget-multisensor tracking. *IEEE Transactions on Aerospace and Electronic Systems*, 50(3):2253–2268, 2014.

Hans Schily, Folker Hoffmann, and Alexander Charlish. A comparison of distributed and centralized control for bearing only emitter localization with sensor swarms. In *SCI-341: Situation Awareness of Swarms and Autonomous Systems*, Tallinn, Estonia (Virtual), 2021.

Michael K. Schneider and Chee Chong. A rollout algorithm to coordinate multiple sensor resources to track and discriminate targets. In Ivan Kadar, editor, *Proceedings Volume 6235, Signal Processing, Sensor Fusion, and Target Recognition XV*, page 62350E, Orlando (Kissimmee), Florida, USA, 2006.

Nicola Secomandi. A rollout policy for the vehicle routing problem with stochastic demands. *Operations Research*, 49(5):796–802, October 2001.

Nicola Secomandi. Analysis of a rollout approach to sequencing problems with stochastic routing applications. *Journal of Heuristics*, 9(4):321–352, 2003.

Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, June 2012.

Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21(5): 1071–1088, 1973.

John R. Spletzer and Camillo J. Taylor. Dynamic sensor planning and control for optimally tracking targets. *The International Journal of Robotics Research*, 22 (1):7–20, 2003.

Tao Sun, Qianchuan Zhao, Peter B. Luh, and Robert N. Tomastik. Optimization of joint replacement policies for multipart systems by a rollout framework. *IEEE Transactions on Automation Science and Engineering*, 5(4):609–619, October 2008.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. MIT Press, Cambridge, Massachusetts, USA, second edi edition, 2018.

Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. *IEEE International Conference on Intelligent Robots and Systems*, pages 4906–4913, 2012.

Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175. IEEE, May 2014.

Gerald Tesauro and Gregory R. Galperin. On-line policy improvement using Monte Carlo search. In *Advances in Neural Information Processing Systems 9*, pages 1068–1074, Denver, CO, USA, 1996.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

Pratap Tokekar, Joshua Vander Hook, and Volkan Isler. Active target localization for bearing based robotic telemetry. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 488–493. IEEE, September 2011.

Pratap Tokekar, Elliot Branson, Joshua Vander Hook, and Volkan Isler. Tracking aquatic invaders: Autonomous robots for monitoring invasive fish. *IEEE Robotics & Automation Magazine*, 20(3):33–41, September 2013.

Olivier Trémois and Jean-Pierre Le Cadre. Optimal observer trajectory in bearings-only tracking for manoeuvring sources. *IEE Proceedings - Radar, Sonar and Navigation*, 146(1):31, 1999.

Marlin W. Ulmer, Justin C. Goodson, Dirk C. Mattfeld, and Marco Hennig. Offline–online approximate dynamic programming for dynamic vehicle routing with stochastic requests. *Transportation Science*, 53(1):185–202, February 2019.

Joshua Vander Hook, Pratap Tokekar, and Volkan Isler. Cautious greedy strategy for bearing-only active localization: Analysis and field experiments. *Journal of Field Robotics*, 31(2):296–318, 2014.

Joshua Vander Hook, Pratap Tokekar, and Volkan Isler. Algorithms for cooperative active localization of static targets with mobile bearing sensors under communication constraints. *IEEE Transactions on Robotics*, 31(4):864–876, 2015.

Sriram Venkateswaran, Jason T. Isaacs, Kingsley Fregene, Richard Ratmansky, Brian M. Sadler, Joao P. Hespanha, and Upamanyu Madhow. RF source-seeking by a micro aerial vehicle using rotation-based angle of arrival estimates. In *American Control Conference (ACC)*, pages 2581–2587, Washington, DC, USA, 2013. IEEE.

Kurt Vonehr, Seth Hilaski, Bruce E. Dunne, and Jeffrey Ward. Software defined radio for direction-finding in UAV wildlife tracking. In *IEEE International Conference on Electro Information Technology*, pages 464–469, Grand Forks, ND, USA, 2016. IEEE.

Matouš Vrba, Jakub Pogran, Václav Pritzl, Vojtěch Spurný, and Martin Saska. Real-time localization of transmission sources using a formation of micro aerial vehicles. In *2019 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 203–208, Irkutsk, Russia, August 2019. IEEE.

Richard G. Wiley. *ELINT – The interception and analysis of radar signals*. Artech House Inc, 2006.

Ramin Zahedi, Lucas W. Krakow, Edwin K. P. Chong, and Ali Pezeshki. Adaptive estimation of time-varying sparse signals. *IEEE Access*, 1:449–464, 2013.

Guoxian Zhang, Silvia Ferrari, and Ming Qian. An information roadmap method for robotic sensor path planning. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 56(1-2):69–98, 2009.

Zi-ning Zhang and Gan-lin Shan. Non-myopic sensor scheduling to track multiple reactive targets. *IET Signal Processing*, 9(1):37–47, 2015.