Otto von Guericke University Magdeburg
Department of Computer Science
Institute for Intelligent Cooperating Systems

Master's Thesis
Compuational Visualistics

DeePolation: AI-based interpolation on multi-dimensional spherical sensors

Author:
Martin Zettwitz

November 06, 2019

Advisors:

Prof. Dr.-Ing. habil. Sanaz Mostaghim
Institute for Intelligent Cooperating Systems

Prof. Dr. rer. nat. habil. Rudolf Kruse
Institute for Intelligent Cooperating Systems

Zettwitz, Martin:
DeePolation: AI-based interpolation on multi-dimensional spherical sensors
Master's Thesis, Otto von Guericke University Magdeburg, 2019.

Abstract

Interpolation is a standard method in computing unknown data. While conventional approaches are designed to find data on known positions between given data, other approaches are designed to optimize function values between given data, i.e., finding the optimal position. Latter is known as the interpolation problem. The problem becomes more complex with higher degrees of the polynomial functions and higher dimensions of the input data. The underlying systems are often overdetermined but can be solved in least-squares sense. The drawback of this method is its limitation to convex functions. Even if the target function is convex, the sampled data may represent it with insufficient accuracy so that extreme points may be missed. Common reasons may be under-sampling by low-resolution sensors. Using high-resolution sensors may overcome this problem, but causes higher computational effort. This work presents a novel solution to this problem for spherical sensors. We overcome the limitation of low-resolution sensors by using Supersampling with deep neural networks to map the low-resolution input to the more precise output of a high-resolution sensor. To prevent pseudo interpolation between local optima, we limit the input data to closed neighborhoods around the original solution. We show practical usage by steering autonomous agents based on spherical sensors with multi-criteria optimization.

Keywords: Interpolation, Optimization, Deep Learning, Supersampling, Spherical Sensor

Acknowledgements

I want to thank all the people that supported me and made this thesis possible. I also want to thank the people that did not believe in me since they pushed my ambition even further. Through the years, some key moments formed the person that I am today. Beginning in high school, where I struggled in math and computer science, my math teacher had faith in me and my skills. At this point, she was right. In addition, I had to thank my father for your tutoring lessons at home, and especially for the faith in me to finish high school. Without you, I would not even be permitted to visit the university. Even after school, or especially after school, you have always been there with supportive advice for all situations in life. You gave me a halt when I doubted. Thank you for always being there without any condition. I also need to thank you, mom, for supporting me during my studies and my whole life. I have to thank my long term friend Martin Kirst. You empowered my curiosity for computer science and gave me the groundbreaking push to work as a computer scientist. But most importantly, you have encouraged me to study computer science after my apprenticeship since you always believed in me. I can't thank you enough for that. A big thank you to all my coworkers from Polarith that always had an open ear during the process of this thesis. I want to thank the department of simulations and graphics, and the institute for intelligent cooperating systems of the Otto von Guericke University Magdeburg for outstanding teaching and supervision. In detail, thanks go to my supervisor Alexander Dockhorn. Starting as my tutor in the bachelor courses, you have become a good friend that always has good advice and supported me not only in this thesis but in life. Special thanks go to Stefan Kirste. I know you for my entire life. After all, we have been through; you are the only one that could always cheer me up and make me smile, even in hard times. No one knows me better than you do. At last, I want to thank all my fellow students and friends. Especially Michi, Alina, Janice, Janos, Doreen, Dome, and Daniel. Together we have been through all the ups and downs during this study and share awesome memories. You have been part of the best time of my life.

Contents

1. Introduction

Autonomous agents are a solid component of modern life. Regardless, they are used as physical robots in real-world applications or as virtual agents in computer simulations, they interact with their environment. To do so, they need to perceive environmental influences, what is done by sensors. In physical applications, for example, with radar, hypersonic, laser, or camera-based sensors. In virtual simulations, sensors can be designed in arbitrary ways since they do not have physical limitations, even though they are often designed to simulate real-world applications. Human-like agents use so-called steering algorithms for their movement. Two major approaches have been proven in practice. First, Classic Steering [Reynolds, 1999], which is computationally fast and easy to use, but is limited in its decision making. Second, the more modern Context Steering [Fray, 2015], which makes a decision based on multiple competing objectives, which is close to real life. The multi-objective problem is solved by Multi-Criteria Optimization (MCO). Especially in Context Steering, multiple solutions are possible thanks to the sensor design. In the basic implementation, solutions can be found only directly at the perceivers of the sensor. This limits the agent's movement domain resulting in unnatural movement by roundabouts. Additionally, movement systems, controllers, respectively, that are not physics-based tend to oscillate, since the movement direction underlies rapid changes. To find better solutions, the resolution of the sensor can be increased, but this comes at high computational costs that are fatal for real-time simulations, especially for crowd simulations. Thus, interpolation schemes are applied to find better solutions around the original one. In terms of movement, the sensor is often circular in 2D setups. Thus, a better solution can be found, based on the decisions on the neighborhood around the original solution by computing gradients and solving a linear equation system [Fray, 2015, Kirst, 2015]. In 3D, the circular sensor becomes spherical. Thus, the distribution of the perceptors is based on a sphere topology, and the neighborhood around a perceptor has no order any more. Additionally, interpolation is more complicated since we now have more neighbors around a solution, resulting in an overdetermined system of linear equations, which is known to provide no solution at all.

Several techniques like least squares regression, Radial Basis Functions (RBFs) or Supersampling exist to solve overdetermined systems, but none of these have been proven as practicable as they are computationally too expensive for real-time simulations, or they are limited in their solution space by design. Thus, new ways for interpolation of spherical sensors must be found with a special focus on the accuracy and computational effort.

1.1 Motivation and goal of the thesis

The goal of this thesis is to design an Artificial Neural Network (ANN) that is able to interpolate, find an optimum, respectively, between given data from a spherical sensor of a MCO based agent. Besides the design of the network, an important focus is on the training setup, which is mandatory for the overall performance of the network. Furthermore, the network needs to be evaluated in terms of accuracy and speed. Thus, meaningful metrics must be found to compare the new method to the current state of the system, especially to a high-resolution sensor that deals as ground truth. First, we focus on the problem in 2D and present a Deep Neural Network (DNN) that is able to interpolate in this environment using a Supersampling approach. Afterward, we expand the problem into the 3D space with light adaptations of the DNN. We show a problem that arises from multiple optima in the environment and how to overcome this problem by limiting the processed input data around the local neighborhood of the original solution. Additionally, we present an advanced layout of the DNN that is able to predict the magnitude of a second objective with respect to the optimized solution of the first objective, that we have predicted before. Finally, we show the limitations of our method and discuss future improvements.

1.2 Structure of the thesis

In the following second chapter, we discuss related work in the fields of steering algorithms, optimization on spheres, advanced interpolation techniques, as well as deep learning-based techniques. The third chapter discusses the background of this thesis and dives deeper into the topics of steering and MCO, interpolation and optimization, and ANNs. In chapter four, we talk about the methodology that has been used in this work. We explain the system setup, the network architecture, the problem and solution of multiple maxima, and how to expand the system for multiple objectives. We show the evaluation of our new method in chapter five, where we first introduce the test setups and, second, present the evaluation results. In the final chapter six, we discuss the results of this thesis and give an outlook on future work and improvements.

2. Related Work

Even though steering Artificial Intelligence (AI) for autonomous agents is well studied due to a large number of applications in robotics, simulations, or games. The most applications, besides individual solutions, use either the principals of Classic Steering [Reynolds, 1999] or Context Steering [Fray, 2015]. Since Classic Steering has benefits for simulating flocks, it has limitations for realistic behavior simulation on individuals, since competing behaviors are mapped on a single solution. Context Steering overcomes these limitations with multiple competing objectives and the usage of a circular sensor with multiple receptors, each yielding a solution. Even though multiple solutions are possible, the best among them must be found to decide for a single solution, which had been handled only in a trivial manner. [Kirst, 2015] presented a complete guideline on how to apply context steering in practice and solved the problem of multiple competing objectives with MCO. The system's accuracy relies on the resolution of the sensor, the number of receptors, respectively. Thus, the accuracy can be increased by a higher count of receptors with a significant increase in high computational costs. [Fray, 2015] and [Kirst, 2015] showed, how to improve the system's decision in terms of accuracy by interpolating an optimal solution based on the gradients of the neighborhood on the sensor. This was presented for the two-dimensional space, while the 3D case has been left for future research. The circular sensor in 2D becomes a spherical sensor in 3D. Hence, the sensor covers even more receptors. Thus, the neighborhood around a decision is larger, and the interpolation problem becomes more complicated. It results in an overdetermined system of linear equations. Hence, more advanced methods for finding an optimum of an unknown function must be applied. Methods like least-squares regression [Trefethen and Bau, 1997] can find a solution but are limited to convex combinations. [Renka, 1984] proposed a method based on triangulation of the sphere but is also limited to convex functions of least-squares regression. [Xu, 2004] showed a solution to the interpolation problem on the unit sphere based on polynomial interpolation, but with restrictions on the position of the sampled points. RBFs [Buhmann, 2003] overcome the limits of least-squares regression, but suffer from high computational

costs for parameterization and searching for a proper solution. [McDonald et al., 2007] showed a fast way of reconstructing data using RBFs with surface response models, but an optimum still needs to be found by an extensive search. Instead of reconstructing the function, one can use techniques based on Supersampling [Crow, 1981], that computes the search space in a higher resolution and interpolates the solution between new data. Since we are not able to compute or optimize in high-resolution space, methods like super-resolution aim to reconstruct the high-resolution signal. Modern approaches [Kim et al., 2016, Ledig et al., 2017] for image data use deep learning for reconstruction from low-resolution images that get close to the original data. Even if we can reconstruct the data in high-resolution space, the problem of finding the optimum still remains. In fact, one could use a high-resolution sensor instead since generating the data is not the bottleneck but the processing. [Du et al., 2018] combined both techniques for image processing to find intensity levels but depends on interaction with user input.

3. Background

3.1 Autonomous Steering Agents

Nowadays, autonomous agents can be found in the real world and simulation scenarios. In industrial simulations, they are used to simulate real-world applications or robotics in advance at low costs to determine problems or optimize workflows before deployment. Autonomous agents are also found in the entertainment industry, like computer games, that are closely related to real-world simulations. Both, in simulations and the real world, they are moved by so-called steering algorithms.

3.1.1 Steering Systems

These kinds of algorithms compute the movement direction and speed, steering angle and acceleration, respectively, of the agents based on environmental influences. While simulations of industrial machines are designed to behave deterministically, human-like agents should be less predictable and act more natural. There are two major approaches to human-like steering.

Classic Steering

The first formulation of human-like steering algorithms, today is known as Classic Steering, was done by Reynolds in the late 1990's [Reynolds, 1999]. The system is based on different behaviors that Reynolds introduced. Each behavior represents a simple task. Very basic behaviors are *follow*, *seek*, and *flee*, which leads the agent directly toward or away from the target as shown in Figure 3.1.

This is done with linear algebra. Each of these low-level behaviors account the position and movement vector of the agent, and the position of the environmental objects, e.g., targets and obstacles. The most simple behavior *follow* is computed in Equation 3.1.

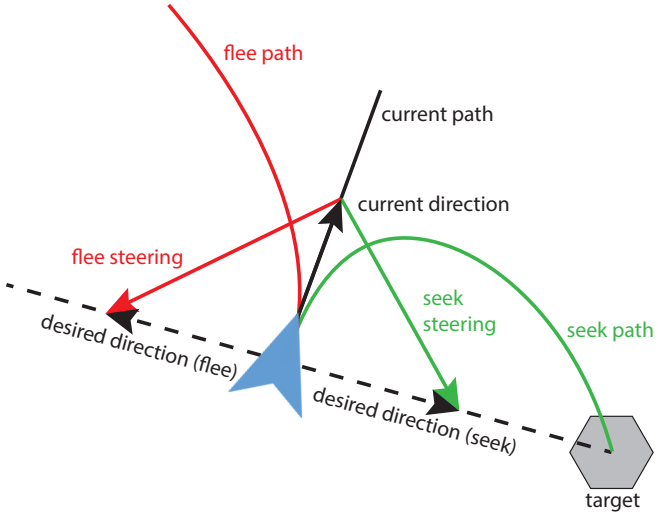


Figure 3.1: *Seek* and *flee* behavior in Classic Steering. The desired direction is based on the position of the agent and the target. The desired direction and current direction result in the (*seek/flee*) steering vectors. Over time, the agent will steer along the new (*seek/flee*) path. Graphic adopted from [Reynolds, 1999]

$$\begin{aligned} \hat{\mathbf{v}} &= \frac{\mathbf{x} - \mathbf{x}_{target}}{\|\mathbf{x} - \mathbf{x}_{target}\|} \\ \hat{\mathbf{x}} &= \hat{\mathbf{v}} - \mathbf{v}, \qquad \mathbf{x}, \hat{\mathbf{x}}, \mathbf{v}, \hat{\mathbf{v}} \in \mathbb{R}^3 \end{aligned} \tag{3.1}$$

Where $\mathbf{x}$ denotes the current position, $\mathbf{v}$ denotes the current velocity in three dimensional space and $\hat{\mathbf{x}}, \hat{\mathbf{v}}$ the updated values. *Seek* is calculated similar but without normalization. To create a greater magnitude when the agent is close to the target, the inverse of $\hat{\mathbf{v}}$ is used. Note that the velocity is optional since the steering angle, direction, respectively, can be computed directly like $\mathbf{v}^*$, and is sufficient to decide for a certain direction. This way, the steering is more general, and the actual movement can be handled by a separate controller that may be physics-based and is specially designed for the particular simulation. A subset of these behaviors, so-called radius steering behaviors, are based on predefined radii, to weight the resulting vector based on the distance. For example, *seek* is scaled by the distance to the target, and *arrive* has a target radius, in which the resulting vector is scaled by the distance to the radius of the target area. A high-level behavior is a weighted linear combination of n low-level behaviors and results in more complex and possibly human-like behavior.

$$\hat{\mathbf{x}}_{total} = \sum_{i=1}^{n} w_i \cdot \hat{\mathbf{x}}_i, \qquad w \in \mathbb{R} \tag{3.2}$$

Since the computations are quite simple and can be done efficiently, it is still widely used in modern simulations and games. The drawback of this method is the risk of

extinguishing behaviors. Since the approach is based on linear algebra and only a single vector is computed for each behavior, the vectors may result in a zero vector. This so-called deadlock can quickly happen if the agent is between two objects with competing behaviors, as shown in Figure 3.2. Hence, the behavior of single agents can be quite unstable and predictable. Thus, Classic Steering is best used to simulate flocks, where the behavior of a single individual is less relevant than the overall behavior of the flock, and the computational costs must be kept low.

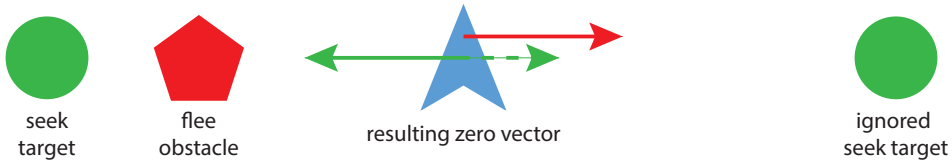
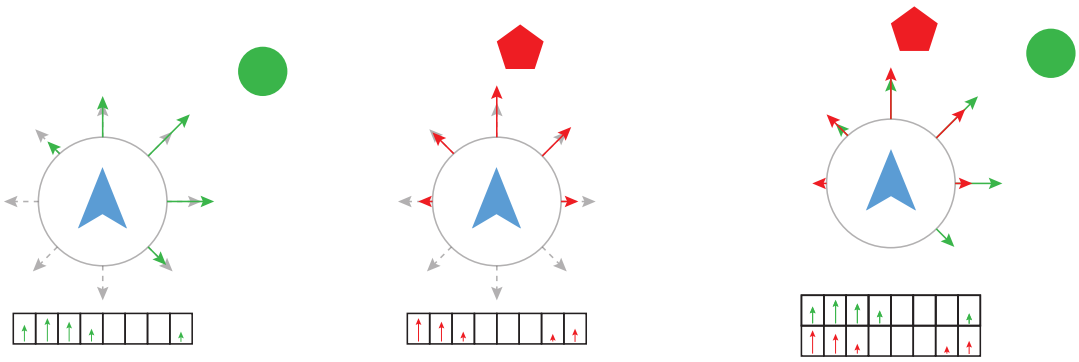


Figure 3.2: The deadlock of Classic Steering. The *seek* and *flee* steering vectors result in a zero vector. The left *seek* vector is greater than the right one since the left *seek* target is closer to the agent. Hence, the left target is preferred, and thus, the right one is ignored. Graphic adopted from [Fray, 2013]

Context Steering

The second approach, so-called Context Steering, was first introduced in a blog post by Fray [Fray, 2013]. He officially published his work two years later [Fray, 2015]. His work is based on the Classic Steering [Reynolds, 1999], but he improved the system's main points of criticism: the limitation to a single decision vector and a single objective function that results in dead-locks. Instead of a single vector with a single solution, Fray introduced a sensor-based system that computes a solution, steering vector, respectively, for each receptor of the sensor, as we can see in Figure 3.3a. We focus on the sensor in more detail in Section 3.1.3. Additionally, Fray expanded the system from a single objective function to multiple objective functions, shown in Figure 3.3c.

In Classic Steering, the solution is computed by combining all vectors to a single vector in a single objective. Hence, only active avoidance is possible by telling the agent to flee from an object actively. In Context Steering, the objects, and thus, the objective functions, have a contextual meaning, like *interest* or *danger*. The objectives are competing. For example, a solution can be of high interest, but also of high danger, and thus, not an optimal choice, like in Figure 3.3c. Context Steering decides for a direction with respect to these objectives. Hence, passive avoidance is possible since objects, directions, respectively, of high danger, are avoided. Based on the sensor, multiple solutions may be valid. Hence, the behavior is more reliable and less predictable, and thus, the agent acts more natural. Fray's work is primarily theoretical as he did not provide particular methods on how to calculate objective values and introduced only a rough method to make a decision based on the objectives. Kirst published a complete guideline [Kirst, 2015] how to apply Context Steering in practice. The objective values are computed based on the Euclidean distance between the target and the sensor, and the angle between the receptor and the target. He solved the problem of competing



(a) Sensor with mapping to an array of *interest* objective

(b) Sensor with mapping to an array of *danger* objective

(c) Comparison of both context maps and the competing objectives

Figure 3.3: A circular sensor with different objectives. The gray arrows mark the original receptors, while the colored arrows represent their magnitude with respect to the target object, objective respectively.

objectives with multi-criteria optimization and made it scalable using the ε-constraint method [Miettinen, 1998]. Hence, the overall behavior has another parameter, threshold, respectively, for each objective. Thus, the agent's behavior can be diversified, e.g., how brave it behaves by moving closer to a dangerous object, solution respectively.

3.1.2 Multi-Criteria Optimization

Since Context Steering handles multiple objectives to make a decision, a so-called multi-objective problem, also called multi-criteria problem, arises. They are defined by multiple competing objectives, where multiple different solutions of equal quality can co-exist. For example, a solution can be of high *interest*, but also high *danger*. Another solution can be of less *danger*, but also of less *interest*, too. An optimization is performed with respect to each objective function f_i, e.g., maximize *interest*, minimize *danger*, which is the default optimization for Context Steering. Due to the curse of dimensionality, the problem becomes more complicated with each additional objective. Such problems can be solved by Multi-Criteria Optimization (MCO), as shown in [Miettinen, 1998]. Therefore, we need to make some definitions, that are put together in Definition 3.1. The decision vectors $\mathbf{x}$ in the solution or decision space $\mathcal{S}$, where $\mathcal{S} \subset \mathbb{R}^n$. The objective functions f_i for m objectives that span the objective space $\mathcal{C}^m$, and the criterion vectors $f_i(\mathbf{x}) \in \mathbb{R}$.

Definition 3.1 (Objective Space)**.** $\mathcal{C} = \{f(\boldsymbol{x}) \in \mathbb{R}^m \mid \boldsymbol{x} \in \mathcal{S}\}$

In Context Steering, the possible solutions $f(\mathbf{x})$ based on the receptor values $\mathbf{x}$ are mapped into the so-called context map $\mathcal{C}$, the objective space, as shown in Figure 3.4. Since many unordered solutions, one for each receptor, exist, the best among them must

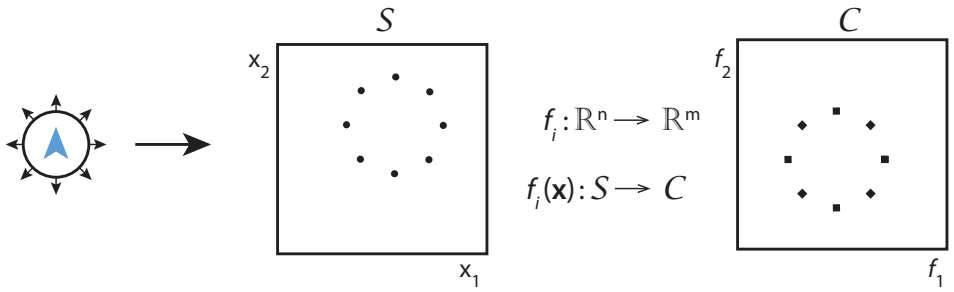
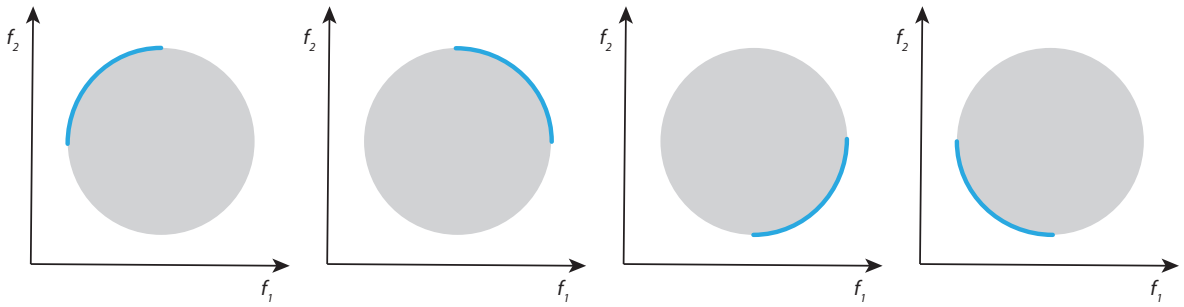


Figure 3.4: Visual representation of the spaces defined in MCO. The decision vectors $\mathbf{x}$ in the decision space $\mathcal{S}$ are mapped into the objective space $\mathcal{C}$ by the objective function f_i.

be found. The best solutions form a subset $s^* \subset \mathcal{S}$. They are mathematically defined as Pareto-optimal or Pareto-dominant, which means that there exists no solution that is better in all objectives, but at least equal or worse in another one. For better understanding w.l.o.g., we focus on minimization: $\min_{\mathbf{x} \in \mathcal{S}} f(\mathbf{x})$.

Definition 3.2 (Pareto Dominance)**.** *A solution* $\boldsymbol{x}$ *is dominant to solution* $\boldsymbol{y}$ *if:*
$\boldsymbol{x} \prec \boldsymbol{y} \Leftrightarrow f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{y}), \quad \forall i \in \{1, ..., m\} \wedge \exists j : f_j(\boldsymbol{x}) < f_j(\boldsymbol{y})$

All Pareto-optimal solutions form a so-called Pareto-front, as shown in Figure 3.5. We still have no single solution. Therefore, different methods can be applied a priori, a posteriori, or interactively. We focus on a priori methods since they are the only unsupervised methods, and thus, suitable for autonomous agents. A graphical overview of the most common a priori methods is shown in Figure 3.6.



(a) $\min(f_1)$, $\max(f_2)$ (b) $\max(f_1)$, $\max(f_2)$ (c) $\max(f_1)$, $\min(f_2)$ (d) $\min(f_1)$, $\min(f_2)$

Figure 3.5: Visual representation of the solutions $f(\mathbf{x})$ in the objective space $\mathcal{C}$. The Pareto-front for different optimization schemes is marked as blue.

Weighted Sum

The objectives are weighted with $w \in [0, 1]$. Thus, the sum over all weighted objectives must be minimized. Note that the sum of all weights must be equal to 1.

$$\min_{\mathbf{x}\in\mathcal{S}} f(\mathbf{x}) = \sum_{i=1}^{m} w_i f_i(\mathbf{x}) \tag{3.3}$$

This method has the drawbacks that the weights must be known in advance, and solutions in concave parts of the front can't be found.

ε-Constraint

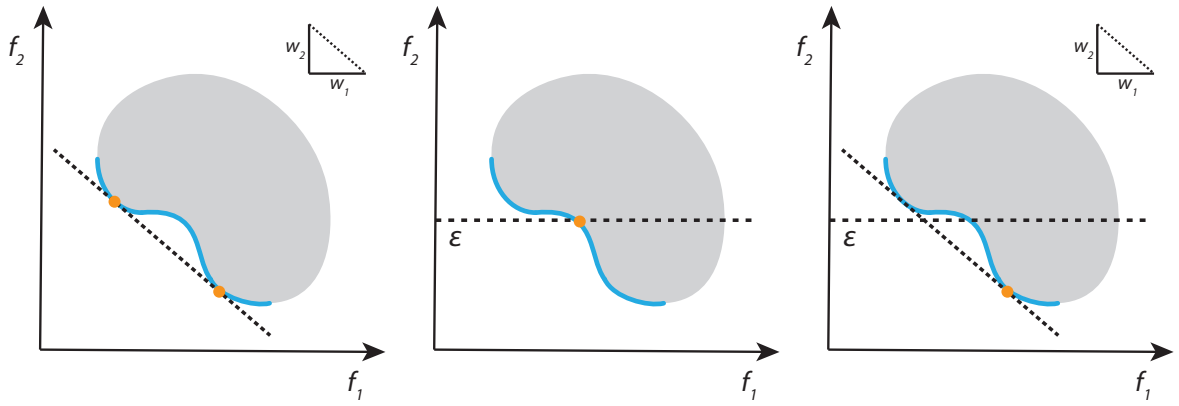
The ε-constraint marks a threshold for an objective, where $\varepsilon \in \mathbb{R}$. The idea is to set a constraint to all but one objective so that there is only a single objective to optimize. All solutions that are above (maximization) or below (minimization) are valid in the sense of the constraint so that the minimization problem becomes:

$$\min_{\mathbf{x}\in\mathcal{S}} f_i(\mathbf{x}), \qquad f_j(\mathbf{x}) < \varepsilon, i \neq j \tag{3.4}$$

Hybrid

This is a combination of both before mentioned methods. First, the ε-constraint is applied to the objective values, and second, a single weighted sum objective function has to be minimized.

$$\min_{\mathbf{x}\in\mathcal{S}} f(\mathbf{x}) = \sum_{i=1}^{m} w_i f_i(\mathbf{x}), \qquad f_j(\mathbf{x}) < \varepsilon, i \neq j \tag{3.5}$$



(a) Weighted Sum (b) ε-Constraint (c) Hybrid approach

Figure 3.6: Visual representation of the most common a priori methods for MCO. The Pareto-front is marked as blue, while the optimal solution is marked as orange.

3.1.3 Sensor Topology

The sensor perceives the environment, and thus, has a significant influence on the creation of the context map. The sensor consists of receptors, that are distributed on the surface of the sensor and point towards its tangent direction. We assume, that the sensor shape is of genus 0, and thus, has a closed surface. Each receptor perceives the environment on its own. The data of all receptors are merged in the sensor. In 2D, a sensor can have arbitrary planar shapes. Hence, the receptors can be distributed equidistantly on the surface, and the neighborhood around a receptor is ordered, as shown in Figure 3.7. Thus, receptors can be traversed in a unique way. In 2D, the most common sensor form for autonomous agents is a circle since every direction can be observed equally. In 3D, the surface is not planar anymore. The receptors position,

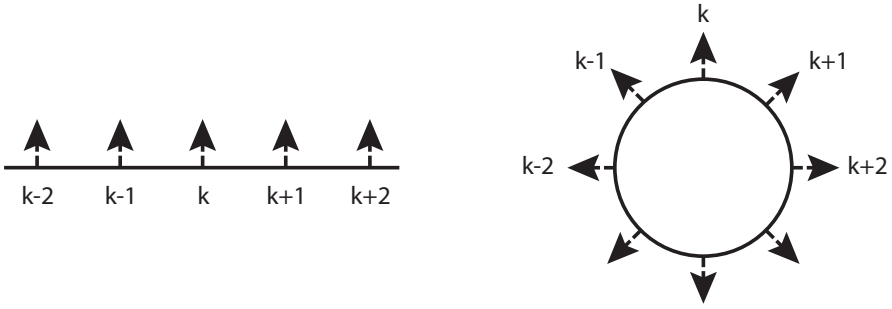
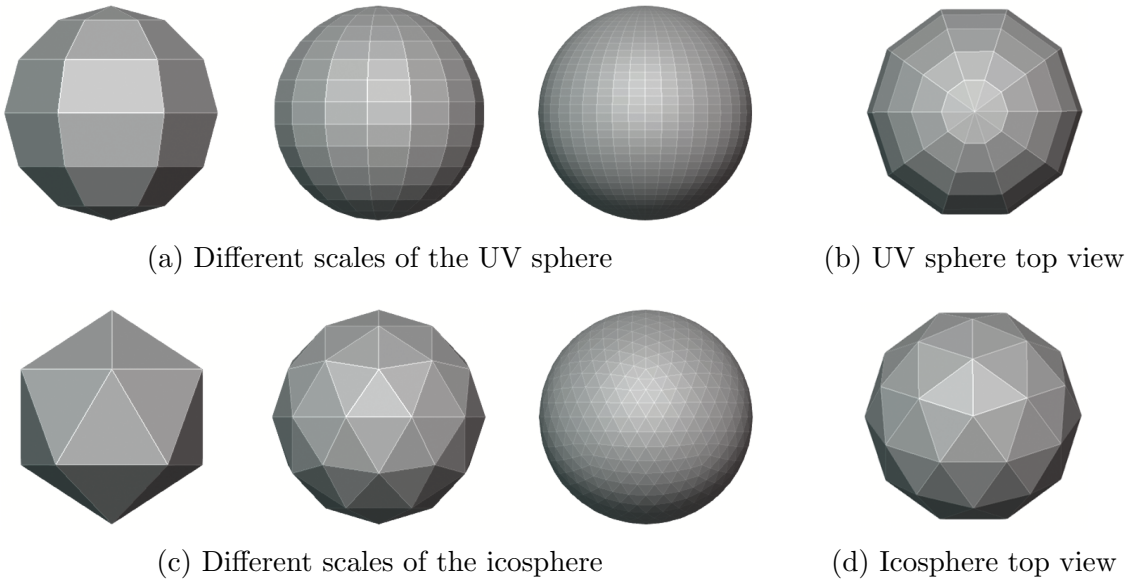


Figure 3.7: Examples for two-dimensional sensors. The neighborhood around receptor k is well defined and ordered.

and thus, the distribution strongly depends on the topology of the sensor. The prior circular sensor becomes spherical, and thus, the neighbors are not ordered anymore since there are more than two in different directions. The most common sphere topologies are the UV sphere and the icosphere. Based on the topology, the receptors are placed on the vertices. The UV sphere is based on UV mapping [Shirley et al., 2009], where U and V denote the axes on the surface, namely the longitude and the latitude. The UV sphere is based on m rings along the longitude and n segments along the latitude. Thus, the sphere consists of $mn - 2n + 2$ vertices as the poles intersect only once. The vertices are arranged in quads, except for triangles at the poles. The UV sphere is well structured with a fixed number of neighbors except for the poles, and is highly scalable, but has the flaw that the vertices are not equidistantly distributed since they converge towards the poles, as shown in Figure 3.8b. The icosphere is based on a geodesic polyhedron [Popko, 2012], especially the icosahedron. Thus the vertices are distributed symmetrically and equally on the sphere. A vertex on the icosphere has six triangles, except for twelve vertices that only have five triangles that arise from the base model without further subdivision, as shown in Figure 3.8d. The icosphere is made of $10n^2 + 2$ vertices, where n is the number of subdivisions. Thus, the number of vertices increases quadratically, and, hence, it is hard to scale. Both topologies are compared in terms of their structure and scalability in Figure 3.8.



(a) Different scales of the UV sphere (b) UV sphere top view

(c) Different scales of the icosphere (d) Icosphere top view

Figure 3.8: Comparison of the UV sphere (a) in different scales and its converging faces at the pole (b), and the icosphere (c) in different scales and its special base vertex with five instead of six neighboring triangles (d).

3.2 Interpolation Methods

Interpolation is the process of estimating new unobserved data between given data of a sampled function. For basic interpolation processes, a regular grid is necessary. Otherwise, advanced interpolation or error minimizing techniques for linear equation systems are needed, as we see in Section 3.2.2. A brief overview of interpolation techniques can be found in [Burger and Burge, 2008]. They can be classified into different types:

- **Point/Area Interpolation**: The coordinates and values of points are known. Interpolation is performed between neighboring data.
- **Global/Local Interpolation**: A single interpolation scheme is applied to a total set or subset of values.
- **Exact/Approximate Interpolation**: The interpolating function(s) exactly fits the data or otherwise approximates it.

3.2.1 Linear Interpolation Methods

Usually, one wants to compute the data at a certain point, e.g., half the distance between values $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$. Therefore, the scalar value t determines the distance or the ratio between $\mathbf{x}_i$ and $\mathbf{x}_j$. $t \in [0, 1]$ is called *interpolation*, while $t \in \{[-\infty, 0), (1, \infty]\}$ is called *extrapolation*. Interpolation is defined in multiple domains and surfaces as a convex combination.

Linear Interpolation

The standard form of interpolation is Linear Interpolation (lerp). There are two ways of computing the new data. First, the gradient between the original values is computed, scaled with t, and added to the first value.

$$\mathbf{x}(t) = \mathbf{x}_0 + t \cdot (\mathbf{x}_1 - \mathbf{x}_0), \qquad \mathbf{x} \in \mathbb{R}^n \tag{3.6}$$

Second, a convex combination weighted with t. This scheme is often used for vectors as in the de Casteljau's algorithm [Farin and Hansford, 2000] to avoid floating-point arithmetic errors.

$$\mathbf{x}(t) = (1-t) \cdot \mathbf{x}_0 + t \cdot \mathbf{x}_1, \qquad \mathbf{x} \in \mathbb{R}^n \tag{3.7}$$

Furthermore, Spherical Linear Interpolation (Slerp) [Shoemake, 1985] is defined by the angle between two vectors on a unit sphere.

$$\mathbf{x}(t) = \frac{\sin\big((1-t)\cdot\theta\big)}{\sin\theta}\mathbf{x}_0 + \frac{\sin(t\theta)}{\sin\theta}\mathbf{x}_1, \qquad \cos\theta = \langle \mathbf{x}_0, \mathbf{x}_1 \rangle, \quad \mathbf{x} \in \mathbb{R}^n \tag{3.8}$$

Bilinear Interpolation

So far, we have been looking at interpolation for functions in 1D. Bilinear interpolation is an extension to interpolate on rectilinear 2D grids. For even higher dimensions, trilinear and other forms of interpolation exist. Bilinear interpolation uses lerp, first in one dimension along the grid cells and again lerp between the two results along the second dimension. A common use-case is a quad mesh.

$$\begin{aligned} \hat{\mathbf{x}}(t_1, 0) &= (1-t_1)\cdot\mathbf{x}_{0,0} &&+ t_1 \cdot \mathbf{x}_{1,0} \\ \tilde{\mathbf{x}}(t_1, 1) &= (1-t_1)\cdot\mathbf{x}_{0,1} &&+ t_1 \cdot \mathbf{x}_{1,1} \\ \mathbf{x}(t_1, t_2) &= (1-t_2)\cdot\hat{\mathbf{x}} &&+ t_2 \cdot \tilde{\mathbf{x}}, \qquad \mathbf{x} \in \mathbb{R}^n \end{aligned} \tag{3.9}$$

Triangular Interpolation

Interpolation is also defined on triangular surfaces, often used for geometric modeling, as shown in [Farin and Hansford, 2000]. Similar to Section 3.2.1, multiple parameters define the interpolation scheme. A point on a triangle is defined as a linear combination of the triangles' vertices $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2$, and the so-called barycentric coordinates u, v, w.

$$\begin{aligned} &u + v + w = 1 \\ &\mathbf{x} = u \cdot \mathbf{x}_0 + v \cdot \mathbf{x}_1 + w \cdot \mathbf{x}_2, \qquad \mathbf{x} \in \mathbb{R}^3, u, v, w \in \mathbb{R} \end{aligned} \tag{3.10}$$

The barycentric coordinates are defined by sub-areas of the triangle and the desired point $\mathbf{x}$.

$$u = \frac{\text{area}(\mathbf{x}, \mathbf{x}_1, \mathbf{x}_2)}{\text{area}(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)} \qquad v = \frac{\text{area}(\mathbf{x}, \mathbf{x}_0, \mathbf{x}_2)}{\text{area}(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)} \qquad w = \frac{\text{area}(\mathbf{x}, \mathbf{x}_0, \mathbf{x}_1)}{\text{area}(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)} \tag{3.11}$$

The area of a triangle can be computed using Heron's formula:

$$\begin{aligned} \text{area}(\mathbf{a},\mathbf{b},\mathbf{c}) &= \sqrt{s(s-a)(s-b)(s-c)} \\ s &= \frac{a+b+c}{2} \\ a = \|\mathbf{b}-\mathbf{c}\| \quad b = \|\mathbf{a}-\mathbf{c}\| \quad c &= \|\mathbf{a}-\mathbf{c}\| \end{aligned} \tag{3.12}$$

If $0 \leq u, v, w \leq 1$ holds true, the barycentric coordinates are a convex combination, and $\mathbf{x}$ is within the triangle (interpolation), else $\mathbf{x}$ is outside (extrapolation). Triangles, and thus triangular interpolation is heavily used in computer graphics and geodesics to model complex surfaces with fast computations. In this work, triangle meshes are used to model the spherical sensor described in Section 3.1.3.

3.2.2 Optimization Methods

Many interpolation problems can not be solved with linear interpolation since the data points are not positioned on a regular grid, or the system of linear equations is underdetermined. In the latter case, regularization in least-squares sense yields a solution. A Lagrange multiplier is often used as a regularization term to add equations to the underdetermined system. High-degree polynomials often interpolate functions of a higher order. The disadvantages of high-degree polynomials are the high computational cost and the high degree of freedom that tend to oscillatory artifacts. To overcome these problems, combinations of piecewise cubic splines, as in Computer Aided Geometric Design (CAGD), are used [Farin and Hansford, 2000]. They represent low-degree polynomials that fit smoothly together to interpolate the function.
Until now, we have been looking at interpolation as a method to find data at a specific position between known data. Without a priori knowledge of the function, the solutions are constrained to the domain of the original data. Interpolation also covers problems of optimization. Therefore, we have to look at the interpolation problem from a different perspective. We no longer want to find the unknown value at a known position, but the unknown position of the maximum (or minimum) value. The new problem is more complicated since we (mostly) neither know the maximum value nor its position or even the exact function. For convenience w.l.o.g., we focus on maxima from now on.

Interpolation Problem

In this work, we want to find the maximum value between two receptors of the sensor described in Section 3.1.3. Therefore, we need to have a look at the neighborhood of the current decided direction, decided receptor, respectively. In 2D, we have a circular or line-shaped sensor. As [Fray, 2015, Kirst, 2015] showed, we need to solve a system of linear equations based on the lines of the 2-neighborhood as one can see in Figure 3.9. There may be a unique solution in case of an intersection, or infinitely many in case the lines are parallel to each other. The latter is only possible in a multi-criteria environment, as shown in Section 3.1.2, where another objective can prevent the system

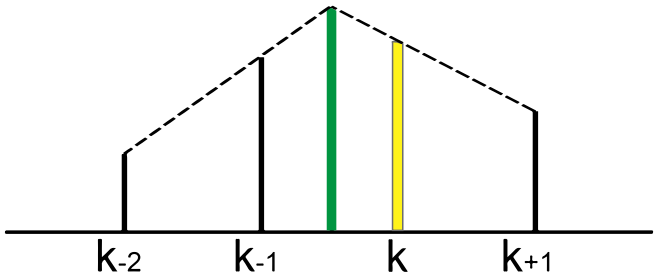


Figure 3.9: Left case of the interpolation in 2D. The slopes of the neighbors k_{-2}, k_{-1} and k, k_{+1} around the original solution k (yellow), intersect in the new maximum (green).

from selecting the maximum value or if all neighboring objectives are perpendicular to the receptors. Otherwise, the current decision is a maximum. Hence the neighboring slopes can not be equal. The general slope-intercept form

$$y = mx + n, \qquad b, m, x, y \in \mathbb{R} \tag{3.13}$$

is adapted to the neighborhood, here w.l.o.g., for the left case$(k_{-2}, k_{-1}, k, k_{+1})$:

$$\nabla f(k) = f(k) - f(k_{-1}) \tag{3.14}$$

$$y = \nabla f(k_{-1}) \cdot x + f(k_{-1}) \tag{3.15}$$

$$y = \nabla f(k_{+1}) \cdot x + f(k) - \nabla f(k_{+1}), \qquad f(k) \in \mathbb{R} \tag{3.16}$$

A new maximum is an intersection. Therefore, Equation 3.15 and Equation 3.16 must be set equal and solved as shown in Equation 3.18 to find the position x, if existent.

$$\nabla f(k_{-1}) \cdot x + f(k_{-1}) = \nabla f(k_{+1}) \cdot x + f(k) - \nabla f(k_{+1}) \tag{3.17}$$

$$\Leftrightarrow x = \frac{f(k) - \nabla f(k_{+1}) - f(k_{-1})}{\nabla f(k_{-1}) - \nabla f(k_{+1})} \tag{3.18}$$

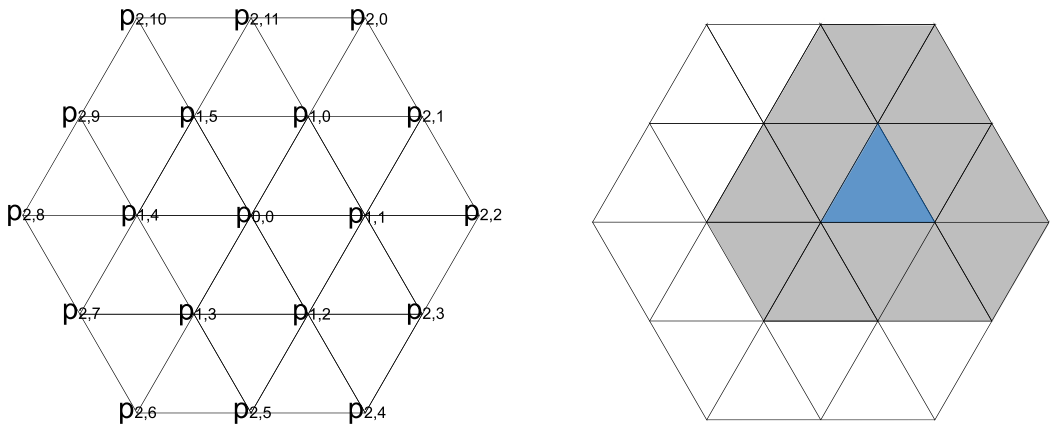
Now, y can be calculated using Equation 3.15. Since we need to consider the right side for the second case, too, the same procedure must be done for $k_{-1}, k, k_{+1}, k_{+2}$. Finally, x correlating with the greatest y value, including the original solution, is taken and the new direction can be interpolated using Equation 3.7 with $t = x$.

The problem is similar in 3D on spherical sensors but does not contain two unknowns and two equations anymore, which makes the 2D system solvable. In 3D, the neighborhood is larger as shown in Figure 3.10a.

A triangle is described by its three vertices $\mathbf{p}, \mathbf{q}, \mathbf{r}$. Since triangles are planar, a plane in its normal form (Equation 3.19) can be described by the triangle vertices, directions, respectively.

$$(\mathbf{x} - \mathbf{p}) \cdot \mathbf{n} = 0 \tag{3.19}$$

$$\mathbf{n} = (\mathbf{q} - \mathbf{p}) \times (\mathbf{r} - \mathbf{p}), \qquad \mathbf{n}, \mathbf{p}, \mathbf{q}, \mathbf{r}, \mathbf{x} \in \mathbb{R}^3 \tag{3.20}$$



(a) 3D 2-neighborhood around centered vertex $\mathbf{p}_{0,0}$

(b) Neighborhood around possible new maximum inside the blue triangle

Figure 3.10: The 2-neighborhood around the centered vertex $\mathbf{p}_{0,0}$ with six neighboring vertices on a triangulated sensor.

For a linear equation system, we need the coordinate form of a plane, described by

$$ax + by + cz = d, \qquad \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \mathbf{n}, \quad a, b, c, d, x, y, z \in \mathbb{R} \tag{3.21}$$

that can be obtained from Equation 3.19. In case of an icosphere as described in Section 3.1.3, we have got six or in special cases five neighboring triangles around the center vertex, as described in Section 3.1.3 and Figure 3.8d. Each of the six, five, respectively, triangles can contain a new maximum. In this topology, a neighborhood similar to the 2D case above, consists of 11 triangles, as shown in Figure 3.10a. This results in an overdetermined linear equation system since we have got more equations than unknowns. Additionally, we no longer have two cases (left and right) but six, five in special cases, respectively, around each of the neighboring vertices to the original solution at the center vertex. One of these six (five) cases is shown in Figure 3.10b. The following example shows how the planes in Equation 3.19 are created in a circular manner inside the 1-neighborhood:

$$\begin{aligned} &(\mathbf{x} - \mathbf{p}_{0,0}) \cdot \big((\mathbf{p}_{1,0} - \mathbf{p}_{0,0}) \times (\mathbf{p}_{1,1} - \mathbf{p}_{0,0})\big) \\ &(\mathbf{x} - \mathbf{p}_{0,0}) \cdot \big((\mathbf{p}_{1,1} - \mathbf{p}_{0,0}) \times (\mathbf{p}_{1,2} - \mathbf{p}_{0,0})\big) \\ &\qquad\qquad \vdots \\ &(\mathbf{x} - \mathbf{p}_{0,0}) \cdot \big((\mathbf{p}_{1,5} - \mathbf{p}_{0,0}) \times (\mathbf{p}_{1,0} - \mathbf{p}_{0,0})\big) \end{aligned} \tag{3.22}$$

and the 2-neighborhood:

$$\begin{gathered}(\mathbf{x}-\mathbf{p}_{1,0})\cdot\big((\mathbf{p}_{2,0}-\mathbf{p}_{1,0})\times(\mathbf{p}_{2,1}-\mathbf{p}_{1,0})\big)\\(\mathbf{x}-\mathbf{p}_{1,0})\cdot\big((\mathbf{p}_{2,1}-\mathbf{p}_{1,0})\times(\mathbf{p}_{2,2}-\mathbf{p}_{1,0})\big)\\\vdots\\(\mathbf{x}-\mathbf{p}_{1,5})\cdot\big((\mathbf{p}_{2,11}-\mathbf{p}_{1,5})\times(\mathbf{p}_{1,0}-\mathbf{p}_{1,5})\big)\\(\mathbf{x}-\mathbf{p}_{1,0})\cdot\big((\mathbf{p}_{2,11}-\mathbf{p}_{1,0})\times(\mathbf{p}_{2,0}-\mathbf{p}_{1,0})\big)\end{gathered}\tag{3.23}$$

Regression

An overdetermined system of the form

$$\mathbf{A}\mathbf{x}=\mathbf{b},\qquad \mathbf{A}\in\mathbb{R}^{m\times n},\mathbf{x}\in\mathbb{R}^{n},\mathbf{b}\in\mathbb{R}^{m},\quad m>n\tag{3.24}$$

where $\mathbf{A}$ is the system matrix consisting of the equations, the unknowns $\mathbf{x}$, and the results $\mathbf{b}$. There is no unique solution, but the system can be solved in least-squares sense by minimizing:

$$\min_{\mathbf{x}}||\mathbf{A}\mathbf{x}-\mathbf{b}||\tag{3.25}$$

$$\Leftrightarrow||\mathbf{A}\mathbf{x}-\mathbf{b}||^2=0\tag{3.26}$$

$$\Leftrightarrow\mathbf{x}=(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}\tag{3.27}$$

To solve this system, one could use the Cholesky decomposition, the QR decomposition, or the Singular Value Decomposition (SVD) as described in [Trefethen and Bau, 1997]. Each yields a solution, but they differ in complexity and numeric stability. While the Cholesky decomposition has a complexity of $\mathcal{O}(\frac{n^3}{3})$ and is the fastest, its condition $\kappa(\mathbf{A}^T\mathbf{A})$ shows numerical instability. The QR decomposition with House Holder Reflections to build its orthogonal base has a complexity of $\mathcal{O}(2mn^2-\frac{2n^2}{3})$. Hence it is computationally more expensive, but its condition $\kappa(A)$ shows that it is numerically more stable. The SVD has the same condition as the QR decomposition. It computes a least-norm solution but comes with a high computational cost of $\mathcal{O}(mn^2-\frac{n^3}{3})$. One problem still remains: least-squares can only solve convex problems. As you can see in Figure 3.9, the desired function values lie outside of the convex hull of the given data. The same holds true for the 3D case. Hence, the result is only a rough approximation of the actual function and yields no reliable solution.

Radial Basis Functions

In contrast to classical regression functions or polynomial functions in general, Radial Basis Functions (RBFs) [Buhmann, 2003] account the samples based on their distance. The closer the sample, the greater its contribution to the function. Therefore, they are appropriate to handle mesh-less data. Their essential component is the basis function ϕ

$$\phi(\|\mathbf{x}\|_2)=\phi(r)\qquad\mathbf{x}\in\mathbb{R}^n,r\in\mathbb{R}\tag{3.28}$$

that maps multivariate functions from $\mathbb{R}^n$ into scalars of $\mathbb{R}$ using the Euclidean norm, and thus is only based on the vector length r which makes it radial. Hence, it is computationally fast, even in high dimensions. The target function $f(\mathbf{x})$ is approximated by the interpolant $h(\mathbf{x})$ as a linear combination of weighted translates of the basis functions $\phi(\|\mathbf{x} - \mathbf{x}_i\|_2)$, where $\mathbf{x}_i$ are called centers:

$$h(\mathbf{x}) = \sum_{i=1}^{n} w_i \phi(\|\mathbf{x} - \mathbf{x}_i\|_2) \tag{3.29}$$

The original function $f(\mathbf{x})$ is approximated as

$$f(\mathbf{x}_j) \approx h(\mathbf{x}_j) = \sum_{i=1}^{n} w_i \phi(\|\mathbf{x}_j - \mathbf{x}_i\|_2) \tag{3.30}$$

The weights w_i can be computed by solving a linear equation system since we have n unknowns and n equations of the matrices

$$\Phi = \begin{bmatrix} \phi(\|\mathbf{x}_1 - \mathbf{x}_1\|) & \dots & \phi(\|\mathbf{x}_1 - \mathbf{x}_n\|) \\ \vdots & \ddots & \vdots \\ \phi(\|\mathbf{x}_n - \mathbf{x}_1\|) & \dots & \phi(\|\mathbf{x}_n - \mathbf{x}_n\|) \end{bmatrix} \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \mathbf{u} = \begin{bmatrix} f(\mathbf{x}_1) \\ \vdots \\ f(\mathbf{x}_n) \end{bmatrix} \tag{3.31}$$

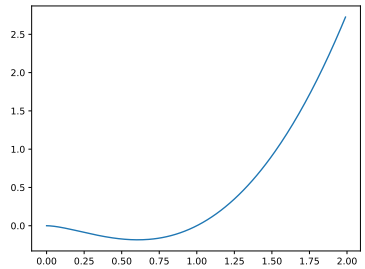
Due to the definition of basic RBFs in Table 3.1 to be positive definite for $(\gamma > 0) \in \mathbb{R}$ and $k \in \mathbb{N}$, the interpolation or kernel matrix Φ is regular. Thus, the system can be solved by inverting Φ or using the LU factorization [Trefethen and Bau, 1997].

$$\mathbf{w} = \Phi^{-1}\mathbf{y} \tag{3.32}$$

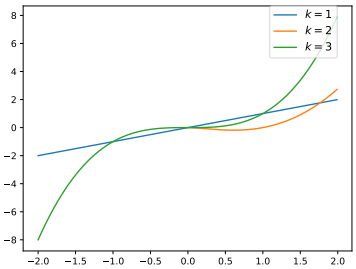
Table 3.1: Most common types of RBFs. They are positive definite for $(\gamma > 0) \in \mathbb{R}$ and $k \in \mathbb{N}$.

Function	$\phi(r)$
Gaussian	$e^{-(\gamma r)^2}$
Multiquadratics	$\sqrt{r^2 + \gamma^2}$
Inverse Multiquadratics	$\frac{1}{\sqrt{r^2+\gamma^2}}$
Thin-plate spline	$r^2 \ln(r)$
Polyharmonic splines	$\begin{cases} r^k & k \in \mathbb{N}_{odd} \\ r^k \ln(r) & k \in \mathbb{N}_{even} \end{cases}$

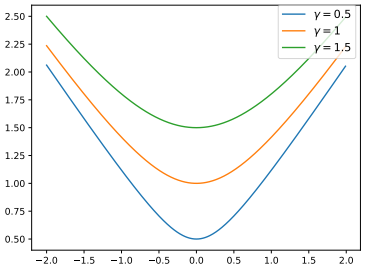
One has to choose an appropriate function for the problem. Since functions like the Gaussian have a parameter, there is another degree of freedom to be fixed. In cases of



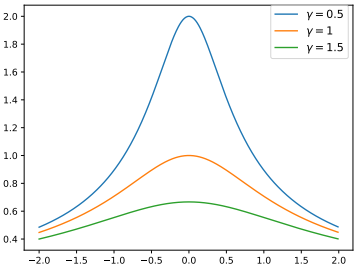
(a) Thin-plate spline



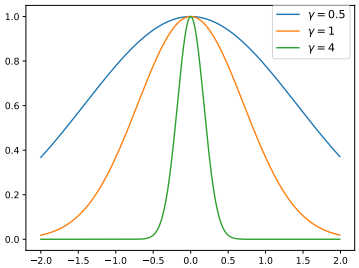
(b) Polyharmonic splines



(c) Multiquadratics



(d) Inverse multiquadratics



(e) Gaussian

Figure 3.11: Overview of the most common RBFs and the influence of their parameters.

Gaussian and Multiquadratic, γ defines the steepness of the function. The greater γ, the more the function oscillates. The smaller γ, the more stable the function is. The influence can be seen in Figure 3.11. γ and other parameters can also be learned by using the Expectation Maximization (EM) algorithm [Dempster et al., 1977]. Therefore, γ is fixed and $\mathbf{w}$ is computed, afterward $\mathbf{w}$ is fixed and γ is computed. The algorithm is repeated until convergence. This way, different parameters for each center can be computed, too. The drawback is the high computational cost that has to be accounted for real-time simulations. After the interpolant $h(\mathbf{x})$ is computed, we still need to find the optimum. This can be done with a great computational effort by a grid search over the solution space within the search domain, or gradient descent. Note that the gradient descent can be stuck in local optima and may not be faster than the systematic search since the step size, learning rate, respectively, may tend to oscillate or converge very slow. Finding the optimum this way, may work for a single or few agents, but is

too expensive for multiple agents in real-time simulations. A more promising approach is to use RBF networks that are closely related to neural networks in Section 3.3.

3.2.3 Supersampling

Prior mentioned methods compute or fit the existing data to interpolate. Supersampling [Crow, 1981] is a technique from the beginnings of computer graphics to overcome aliasing artifacts. The original problem arises from computing, sampling, respectively, an image or a scene for rendering. On edges, the visually appealing color varies as the visualized pixel may lay between two different textures. Supersampling computes the image in a higher resolution than the output resolution and samples at different positions around the original pixel in high-resolution space. Hence, the new pixel value is averaged, interpolated, respectively, and thus, more accurate. Even though the accuracy is highly scalable, the drawback of this technique is the high computational cost for rendering in high-resolution space. Adapted to the problem of interpolation on a sensor, we have the limitation that we are not able to compute data in higher dimensions right away. We can obtain high-resolution data only by the usage of a high-resolution sensor, which makes the whole process ambivalent. So-called Super-Resolution methods perform upscaling of low-resolution data intending to reconstruct the data in high resolution. Modern approaches [Ledig et al., 2017, Kim et al., 2016] use deep learning for upscaling without much loss of the original data, but the problems of expensive computations in the high-resolution domain and finding the optimum still remain. In this work, we propose a novel method to compute the optimum of a low-resolution sensor without the high cost of Supersampling or pre-used Super Resolution but based on their principles of obtaining high-resolution data.

3.3 Artificial Neural Networks

Artificial Neural Networks (ANNs) [Kruse et al., 2016, Goodfellow et al., 2016] cover a subset of stochastic machine learning methods. Their design is motivated by structures of the human brain. Thus they are a complex network of single neurons. In the past decade, ANNs became very popular due to the available computational power and potential of massive parallelization on modern graphic cards, the availability of powerful backends [Abadi et al., 2016], and easy to use high-level Application Programming Interfaces (APIs) [Chollet et al., 2018]. Even though ANNs need much time for training, the final network predicts very fast.

Basics

As mentioned before, ANNs consist of connected neurons. A basic neuron has m inputs, a bias value θ, and n outputs. They are connected via the inputs and outputs, which are weighted with w. A neuron is activated, if the weighted sum of the input exceeds the threshold θ. If activated, its output is computed with an activation function. Neurons are organized in layers. The first one is the input layer that is connected to the data

that are passed through the network. Since the number of neurons is fixed, the number of input neurons must match the number of input variables. The following layers are so-called hidden layers since they are not visible to the user, and their values are optimized during the learning process. The last layer is the output layer with neurons depending on the problem and the output that should be predicted. Layers are called fully connected if each neuron is connected to each neuron of the previous layer. ANNs, where the data is passed unidirectionally from the input to the output, are called feed-forward networks. Other forms, like recurrent neural networks, where neurons are linked to preceding neurons, exist, but are not be discussed in this work. Their potential to solve problems depends on the topology, and primarily the size and amount of the layers — the more neurons in a layer, the greater its ability to adapt to complex problems. The same holds true for the number of hidden layers. In terms of multiple hidden layers, we speak of Deep Neural Networks (DNNs) or deep learning. While the first layers separate low-level information or features, the deeper layers connect them to more meaningful information like patterns. The learning process is mostly supervised by making predictions that are based on the input, and are compared to ground truth data.

Activation Functions

The activation function σ influences the output of a single neuron. The output of σ depends on the output a of the connected neurons from the previous layer, the weights w for these connections, and the bias θ. The output a for neuron j in layer i with m input connections is computed as a weighted sum:

$$\begin{aligned} a_{i,j} &= \sigma(w_{j,0} \cdot a_{i-1,0} + \cdots + w_{j,m} \cdot a_{i-1,m} + \theta_j) \\ \Leftrightarrow a_{i,j} &= \sigma(\sum_{k=0}^{m}(w_{j,k} \cdot a_{i-1,k}) + \theta_j) \end{aligned} \tag{3.33}$$

Since the weighted sum for a single neuron a can be written as a dot product, the output for all l neurons inside a layer can be expressed as a matrix-vector product. Thus, the column vector $\mathbf{a}$ is the output for all neurons in a layer, $\mathbf{W}$ denotes the weight matrix, and Θ is a column vector containing the biases.

$$\begin{aligned} \mathbf{a}_i &= \sigma\left(\begin{bmatrix} w_{0,0} & \dots & w_{0,m} \\ \vdots & \ddots & \vdots \\ w_{l,0} & \dots & w_{l,m} \end{bmatrix} \begin{bmatrix} a_{i-1,0} \\ \vdots \\ a_{i-1,m} \end{bmatrix} + \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_m \end{bmatrix}\right) \\ \Leftrightarrow \mathbf{a}_i &= \sigma(\mathbf{W}\mathbf{a}_{i-1} + \Theta) \end{aligned} \tag{3.34}$$

The matrix representation comes with a large potential for numerical fast computations and parallelization, e.g., using block matrices. For activation functions, it is important that they do not saturate and are differentiable. Latter is mandatory to apply the back-propagation algorithm for learning. If they do saturate, especially towards their

boundary, the input values must become extremely large to increase the function value, extremely small to decrease, respectively. Since the function has a strong influence on the gradient using the back-propagation algorithm, it is important that high values result in high gradients. Two problems arise from this context. First, the vanishing gradients, where the gradients are in the range of [0,1], and thus, become smaller and smaller by each propagation through the network layers. Second, the inverse problem of exploding gradients, where the gradients become larger and larger, and thus, result in an unstable behavior or numerical overflow. The most important activation functions are shown in Table 3.2, while Rectified Linear Units (ReLUs) [Glorot et al., 2011] and Exponential Linear Units (ELUs) [Clevert et al., 2015] are mostly used for hidden layers since they do not saturate. For the output layer, the most used activation functions are softmax for exclusive categorical problems, sigmoid for non-exclusive categorical problems, and the identity function for unbound outputs.

Table 3.2: Most common types of activation functions for ANNs.

Function	$f(x)$
Identity	$f(x) = x$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
Hyperbolic Tangent	$f(x) = \tanh(x)$
Softmax	$f(x) = \ln(1 + e^x)$
ReLU	$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$
ELU	$f(x) = \begin{cases} \alpha(e^x - 1) & x < 0 \\ x & x \geq 0 \end{cases}$

Cost Functions

Feed-forward networks are trained by stochastic gradient descent with the back-propagation algorithm using a cost function $C(\mathbf{W}, \Theta, \mathbf{x}, \mathbf{y})$, that represents a metric. Where $\mathbf{x}$ are the input samples and $\mathbf{y}$ are the desired outputs, ground truth data, respectively. The back-propagation algorithm computes the error between the ground truth $\mathbf{y}$ and the prediction based on the cost function C, and propagates it recursively through the network by distributing the error on the network's parameter based on their influence on the error. To train the network, the influences of the weights and biases for the whole network are computed. Since the output a_i of a neuron in Equation 3.33 is based on the weights w, the preceding neurons a_{i-1}, and the bias θ, the influence for each of these components can be computed by the gradient with respect to the error. Therefore, we must compute the partial derivatives of the parameters and apply the chain rule. This

procedure is repeated recursively from the last layer to the first layer. Additionally, the error function is averaged over a batch of samples for faster convergence. After each epoch, the weights and biases are adjusted. Since we compute gradients, derivatives, respectively, it is mandatory that the cost and activation functions are differentiable. The cost function is highly specific for the type of problem that should be solved by the ANN. Basic cost functions are the mean squared error, mean absolute error, binary cross-entropy, categorical cross-entropy, and the cosine proximity.

Optimization

Neural networks can be optimized in their performance since we aim for the network's ability of generalization, so that, the network is not able to predict only before seen training examples but unknown data. Hence, a massive amount of training examples is mandatory to learn as much different information as possible. Augmentation methods are used to improve the amount of available training data. The network can be further optimized by the number of nodes per layer and the number of layers. Since deep networks achieve similar performance to thin and large networks, but with less computational effort, deeper networks are preferred. The amount of neurons has a significant influence on the performance of the network. While too few neurons inhibit the network from fitting the training data well, too many neurons fit the data too close. In the former case, we speak of underfitting; in the latter, we speak of overfitting [Hastie et al., 2009]. The goal is to find a configuration to get the best performance with the least amount of neurons so that the network is fast, able to generalize, and achieves a good performance. To reach this goal, one can use a bottom-up or a top-down approach. In the former case, the neurons are initially high and are decreased until the performance becomes poor. In the latter case, the number of neurons is increased until a good performance is reached. To detect under- or overfitting, the data is split into training, validation, and test sets. Thus, overfitting becomes visible by high losses and a gap between the training and validation accuracy. Underfitting becomes visible by a poor accuracy on the training set. Additionally, a method called Dropout Regularization [Srivastava et al., 2014] is used to prevent overfitting, and to improve the network's ability to generalize. Dropout Regularization uses a new so-called Dropoutlayer, which randomly sets $\alpha \in [0, 1]$ percent of a layers total inputs to zero during training. Thus, the following layer learns a sparse representation of the previous layer. Since gradient descent can be stuck in a local optimum, the learning rate has an influence on the performance, too. Too small learning rates converge very slow and have a high risk of getting stuck even in small local optima. Large learning rates tend to oscillate, and thus, may never converge. Thus, the learning rate controls the influence of the gradient. Modern approaches like Adagrad [Duchi et al., 2011] or Adam [Kingma and Ba, 2014] use an adaptive learning rate.

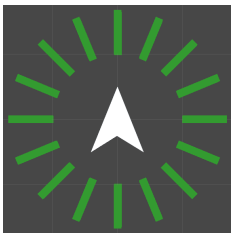
4. Methods

In this chapter, we talk about our novel method in detail. In short, we design a DNN that takes the low-resolution context map of a MCO-based agent as input. We limit the context maps to the neighborhood around the original solution of the MCO solver to prevent anomalous interpolation. The decision of the same system with a high-resolution sensor is used as the training target. Thus, the network is trained to map from low-resolution input to high-resolution output.

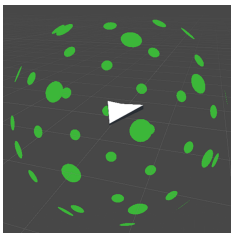
4.1 System Setup

4.1.1 Agents

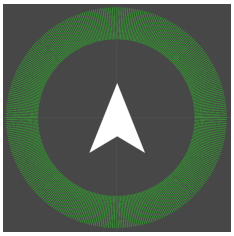
We use agents that are based on Context Steering, as described in Section 3.1.1, with a MCO solver under the usage of the ε-constraint method. A single objective needs to be maximized, while all others need to be minimized. This is close to real-life scenarios or simulations where the agent should move towards a target (maximize) and evade obstacles (minimize). Based on the context map, the system computes a decision in the form of a movement direction, receptor Id, respectively. The agents have a circular or spherical sensor with regularly spaced receptors. The spherical sensor is based on an icosphere. As one can see in Figure 4.1, they are attached in different resolutions. This way, we obtain correlating input and output data of the same time step and the same environment. While the low-resolution sensors are used to create the input data for the learning process, the high-resolution sensors are used to create the target output. Thus, we can combine a precise output direction with a sparse context map, similar to Supersampling. Since the decision of the MCO solver is independent of the attached behaviors but relies on the context map, we use the *seek* behavior as described in Section 3.1.1 to build the context map based on the distance and the angle relative to the target. Thus, we can create values in the full normalized range between [0,1], which is mandatory for the DNN to generalize. The attached controller is based on simple



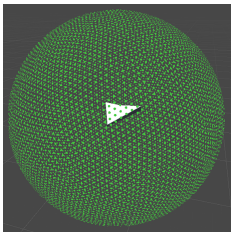
(a) Circular sensor: low resolution



(b) Spherical sensor: low resolution



(c) Circular sensor: high resolution



(d) Spherical sensor: high resolution

Figure 4.1: Comparison of the low-resolution circular sensor with 16 receptors (a), the low-resolution spherical sensor with 42 receptors (b), the high-resolution circular sensor with 720 receptors (b), and the high-resolution spherical sensor with 10242 receptors (d).

translations and rotations since the influence of the interpolation is here much better visible. Without interpolation, the agent tends to oscillate between two receptors or performs harsh jumps between them.

4.1.2 Training Data

Instead of actual simulation scenes from games, we use laboratory scenes since the context map and the principle of the MCO solver are independent of the scene. We place an agent in the origin of the scene and disable the controller, so that, the agent remains static. This is necessary as the agent would always stay close and oriented toward the target, causing a falsified context map. Now, we move the target objects around the agent systematically. We talk about the target's movement in more detail in Section 4.2.2 and Section 4.2.3. The state is evaluated every 20ms. Thus, the movement speed of the targets must be set properly. In every update, the current state for both sensors is written into an external data structure that can be seen in Table 4.1. In the base setup, the whole context map is stored inside the external data. Thus, the network learns the maximum value, which represents the decision, based on the complete context map. For efficiency reasons, we store only the complete context map for the low-resolution sensor since the context map for the high-resolution sensor is extremely large and not necessary for the training. Thus, the processing during the generation is faster, and the memory consumption is significantly lower. The training data are split into a training, validation, and test set in the ratio of 80:13:7.

Table 4.1: The state data of the MCO agent.

Component	Description
Context Map	The values of each receptor for an objective
Decided Direction	The decided direction in $\mathbb{R}^3$
Decided Values	The values of the decided receptor for an objective
Receptor Count	The number of receptors that are attached to the sensor

4.1.3 Technical Setup

From the technical sight of view, we use the Unity Game Engine [Unity Technologies, 2019] for the simulation and Polarith AI [Polarith UG, 2019] for the autonomous agents with Context Steering. The state of the agent is stored in a JSON file, which is processed by the learning system. The learning system contains the preprocessing of the training data, and the neural network. The backend of the DNN is Tensorflow [Abadi et al., 2016] that runs on a CPU since the network architecture is compact, and thus, there is no benefit of using a GPU and its overhead. The final network is used to predict the direction based on the input of the current context map. Therefore, both systems, Unity, and Tensorflow, must communicate with each other. For this purpose, we use UDP as a fast communication protocol.

4.2 Basic Network Architecture

4.2.1 Topology

First, we interpolate only a single objective since the primary objective determines the decided direction. The basic network has a simple topology for both, 2D and 3D. It consists of the input layer with neurons according to the size of the context map, low-resolution sensor, respectively. The hidden layers use ReLU as activation function since the input from the context map is always positive. They are fully connected with a decreasing number of neurons per layer, resulting in a tapered shape. Let n be the total number of neurons in the first layer $i = 0$ and m the total number of hidden layers. Each consecutive hidden layer $i > 0, i < m$ has $(1 - i/m)n$ neurons with a minimum of three neurons. After the first hidden layer, a Dropoutlayer is added. The output layer consists of three neurons with a linear (identity) activation function for a continuous output since we predict a directional vector in $\mathbb{R}^3$. We use the mean-squared-error loss function as the network output is continuous, and thus, directly correlates with errors. Alternatively, the interpolation problem can be seen as a classification problem, where each output category refers to an Id of the high-resolution sensor. Thus, a look-up table is used to determine the corresponding direction. Note that the network would

be significantly larger because the output layer scales with the discrete high-resolution sensor. Additionally, a more complex loss function is needed since a false category in the network domain may be close to the correct solution in the spatial domain. In contrast, the proposed vector variant is independent of the high-resolution sensor, and is continuous, too. A complete overview of our network topology is given in Figure 4.2.

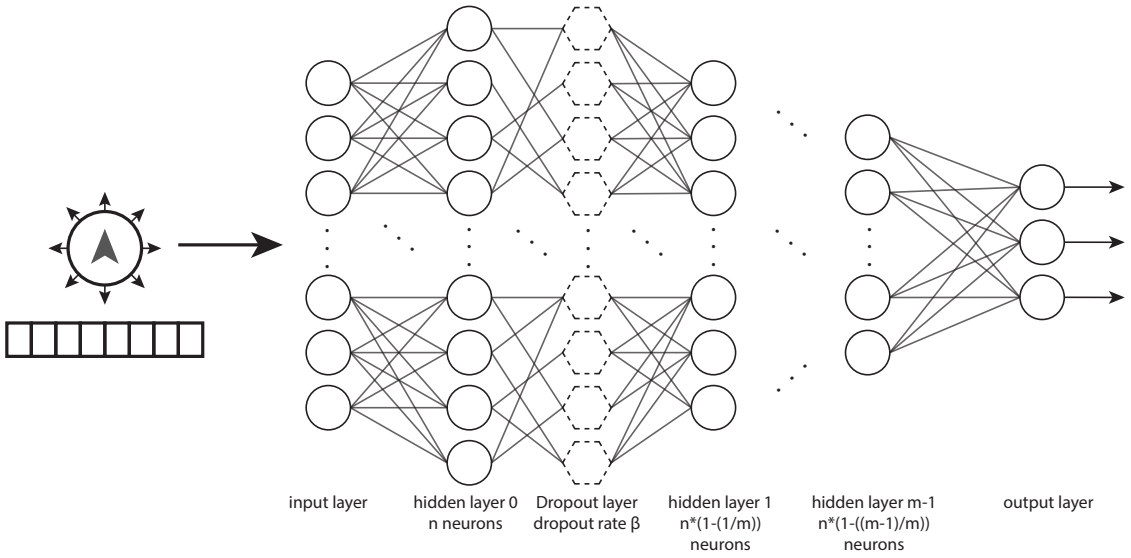


Figure 4.2: The topology of the basic network. The size of the input layer scales with the sensor resolution. The fully connected hidden layers consecutively have fewer neurons, resulting in a tapered shape. A Dropout layer prevents overfitting. The output layer has three neurons to predict a vector in $\mathbb{R}^3$.

4.2.2 2D Environment

At this point, we talk about the specific network implementation and training setup of the two-dimensional case. Note that we didn't perform an optimization on the hyperparameters as the network's only purpose is to demonstrate the ability to solve the interpolation problem in 2D. Thus, it is only a proof of concept.

Network Architecture

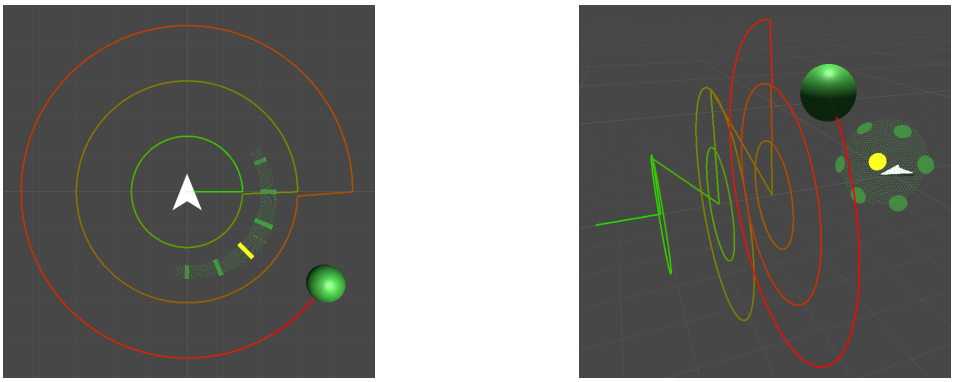
The topology refers to Section 4.2.1. As the agent's sensor consists of 16 receptors, the context map has 16 entries, and thus, the network's input layer has 16 neurons, too. Due to the fact that the 2D interpolation problem is solvable with linear equations, the network is simple, so we only need a single hidden layer consisting of $n = 30$ neurons. The Dropoutlayer has a dropout rate β of 10%. We are able to achieve an accuracy of 97.02% and 0.02 loss on the test set. Figure 4.4a shows the interpolation of the network in practice.

Training Setup

As explained in Section 4.1.2, the agent is placed in the origin and remains static. A single target of the objective *interest* is placed in the origin with a small offset, too. Since the sensor is circular, the target needs to perform a circular movement around the agent to create values of *interest* in the context map equally. We attach a *seek* behavior with an inner radius r_i and outer radius r_o, and inverse linear distance mapping. Thus, the closer the target, the greater the *interest* value. To ensure that every receptor receives values in the normalized domain of [0,1], the radius r between the target and the agent must increase after every full rotation in the range $[r_i, r_o]$. Additionally, the angle speed must be constant to prevent bias. We set a time period of 2 seconds to perform a full rotation. According to Unity's update interval of $\approx$ 20ms, we receive values every 3.6 degrees. The radius r is increased after every full rotation by

$$\hat{r} = r + (r_{outer} - r_{inner}) \cdot \gamma \tag{4.1}$$

where $\hat{r}$ denotes the updated radius and $\gamma \in [0, 1]$ is set to 0.05. In total, we create $\approx$ 2500 training samples. The training setup is shown in Figure 4.3a.



(a) Trajectory of the 2D training

(b) Trajectory of the 3D training

Figure 4.3: The training setups of the basic network. The trajectories show the rotation path of the target, which is rotated around the agent. Therefore, the radius r is adjusted after every full rotation. In 2D (a), the target moves in circles around the agent, while in 3D (b), the target moves in a spherical manner.

4.2.3 3D Environment

Similar to the 2D case above, this early version serves as a proof of concept, and thus, we did not optimize the hyperparameters.

Network Architecture

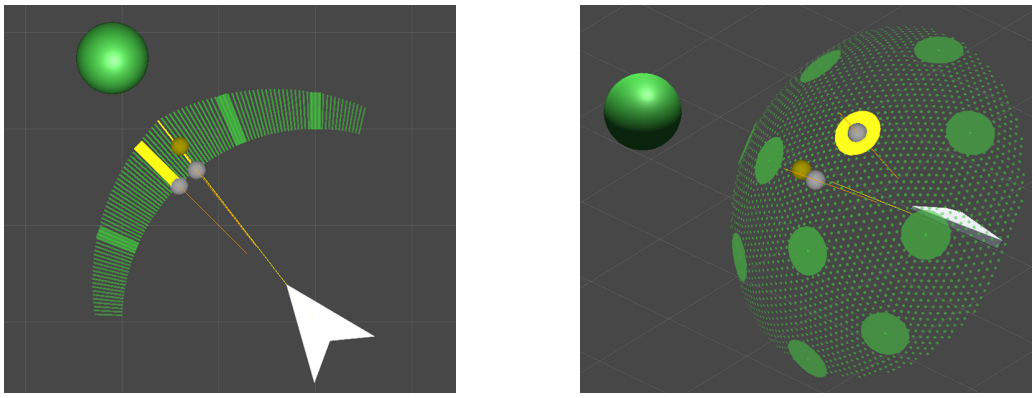
The spherical sensor has 42 receptors, and thus, the input layer has 42 neurons as well. Since the 3D problem is more complex, we use two hidden layers with $n = 40$ neurons in the first, and 20 neurons in the second layer. The Dropoutlayer again has a dropout rate of 10%. Similar to the 2D network, we achieve an accuracy of 98% with 0.01 loss on the test set. The interpolation based on the trained network is shown in Figure 4.4b.

Training Setup

Due to the fact that we must cover an additional dimension, the test setup of Section 4.2.2 needs to be expanded. Therefore, the circular movement of the target is translated along the x-axis, while the rotations are performed in the yz-plane. The rotation radius r is limited by the distance to the agent. Hence, we introduce another radius r_t for the translation along the x-axis in the range of $[-r_o, r_o]$. r_t is updated like r in Equation 4.1, after r has reached its current maximum, that depends on r_t.

$$r_{max} = r_o - \mid r_t \mid \tag{4.2}$$

As a result, the rotations approximate a sphere and we ensure that each receptor receives the same values. We create about 27000 samples in total. The training setup, including the trajectory of the target, is shown in Figure 4.3b.



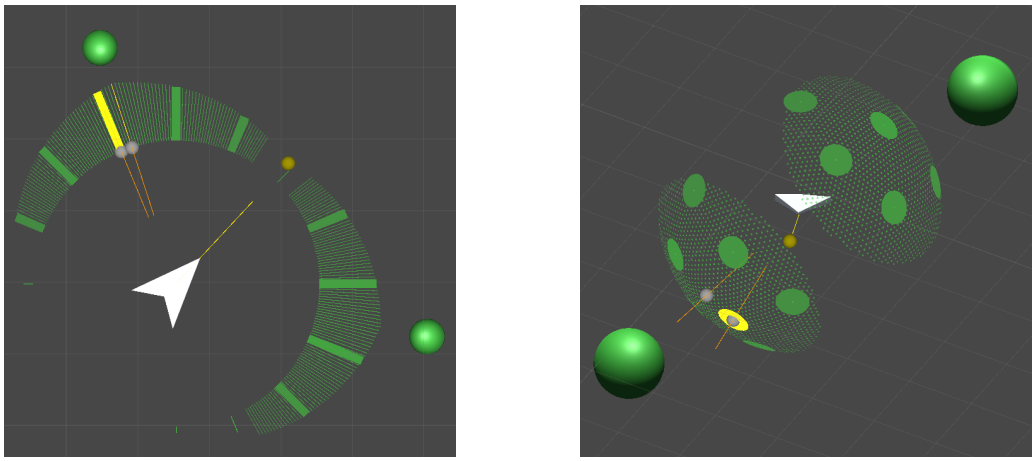
(a) Interpolation in 2D (b) Interpolation in 3D

Figure 4.4: The results of the simple network architectures. The original low-resolution solution is marked with a yellow bar, dot, respectively. The gray spheres and orange lines mark both, the low, and the high-resolution solution. The golden sphere and golden line mark the interpolated solution. Note that the interpolated solution and the high-resolution solution are nearly collinear.

4.3 Multiple Maxima

The approach above works well for a single target object but yields wrong results when applied to setups with multiple target objects. As one can see in Figure 4.5, the network's decision, marked with a golden sphere, is between both maxima. We assume that the network averages the decision for both maxima. We can solve the problem in two ways. First, we can adjust the network itself, or second, we can alter the training setup for multiple targets. Since we can not make sure to cover all possible situations for multiple maxima, we decide to change the network and the input data. Thus, we limit the context map to a k-neighborhood around the original decision, as described in Section 3.2.2 and Figure 3.10, where k denotes the hops to a certain

receptor. Interestingly, the original 3D network tested with a single target and a limited k-neighborhood, yields unstable results for $k < 4$. We expect k to be smaller than 4 since a 2-neighborhood is sufficient for a solution, and $k = 4$ covers more than half of the sensor. The neighborhood for different k, and the resulting interpolator are shown in Figure 4.6.



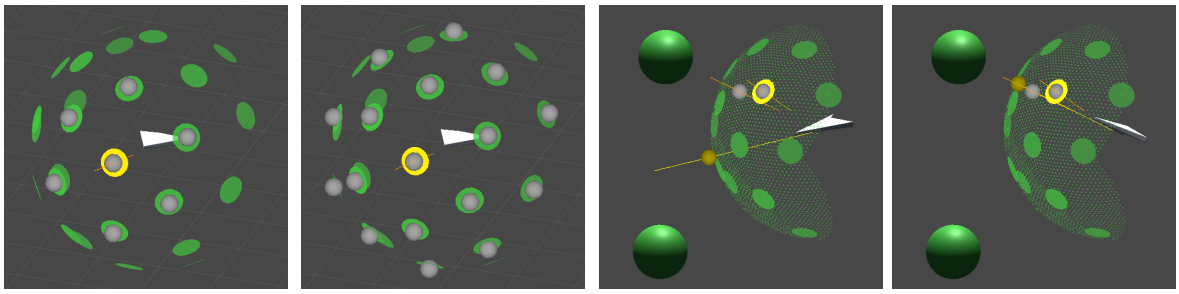
(a) Anomalous interpolation in 2D (b) Anomalous interpolation in 3D

Figure 4.5: Due to multiple maxima in the scene, the network interpolates between both of them. In 2D (a), as well as in 3D (b), the interpolation tends to averaging.

We change the network architecture for the adjusted training samples with a neighborhood of $k = 1$, and use the same training scenario with a single rotation target. We found $m = 4$ hidden layers, with a maximum of $n = 140$ neurons, a dropout rate of $\beta = 10\%$, Adam as optimizer for the gradient descent, a normal distribution for the weight initialization, and 10 epochs with a batch size of 80 to be optimal. Thus, we achieved an accuracy of 97.43% and 0.03 loss on the test data. The interpolation is quite accurate, as shown in Figure 4.6d.

4.4 Multiple Objectives

The methods above show a solution for the interpolation problem with a single objective. Since Context Steering handles multi-objective problems, we need to find the value for the other objectives as well. If the newly interpolated solution violates a constraint, another solution must be found. Optimal, the best solution without violating the constraints, is chosen. Unfortunately, this is a very complex problem since the objectives span multiple surfaces that intersect each other. The available solutions with respect to the constraints may be disjunctive surfaces, and thus, not unique. A trivial solution to this problem was proposed by [Kirst, 2015], where the interpolated solution is discarded if a constraint is violated, and thus, the valid original solution is used instead. Since we interpolate only the first objective, the others must be computed as well, to check if the constraint is violated at the new position. Therefore, multiple approaches are possible.



(a) Neighborhood for $k = 1$ (b) Neighborhood for $k = 2$ (c) Interpolation without neighborhood (d) Interpolation with $k = 1$

Figure 4.6: Different k-neighborhoods around the original solution, which is marked as yellow. The k-neighborhood around the original solution is marked with gray spheres. (a) shows the neighborhood for $k = 1$, while (b) shows the neighborhood for $k = 2$. (c) shows the interpolated result for multiple maxima without a neighborhood, while (d) uses a neighborhood of $k = 1$.

4.4.1 Linear Interpolation

As a fast and simple solution, we can approximate the other objectives f_i by using linear interpolation. Note that due to the nature of linear interpolation, we can only compute values that are in the range between the original vertices. Since we need to interpolate on triangles, we need to use barycentric coordinates. Therefore, we must determine the adjacent vertices that span the triangle, which covers the interpolated solution, as one can see in Figure 4.7a. We must project $\mathbf{x}$ onto the plane that is spanned by the triangle to ensure they are coplanar. The vertices $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^3$ are normalized to compute the barycentric coordinates u, v, w for the normalized projected position $\mathbf{x}$ with Equation 3.11. a_i, b_i, c_i are the corresponding magnitudes of the non-normalized vertices, receptors, respectively. Now, we can compute the objective value x_i by inserting a_i, b_i, c_i in an altered version of Equation 3.10.

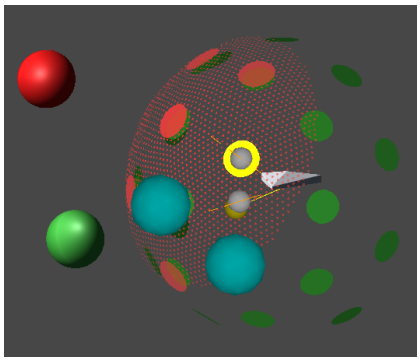
$$x = u \cdot a + v \cdot b + w \cdot c, \qquad a, b, c, u, v, w, x, \in \mathbb{R} \tag{4.3}$$

4.4.2 Advanced Network Architecture

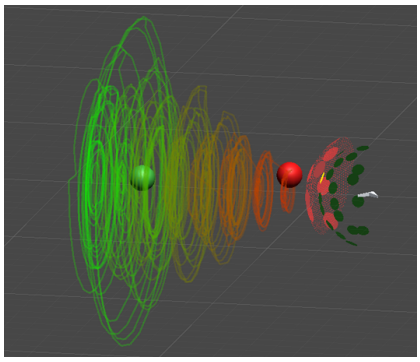
To overcome the limitations of linear interpolation, we expand our neural network to compute the values of the second objective at the position of the interpolated solution.

Topology

Therefore, we separated the network into two inputs and two outputs, one for each objective. The first part of the network takes the context map of the first objective as input and computes the interpolated solution as done before in Section 4.3. The other part combines the inputs of both objectives, context maps, respectively, using a merge layer to compute the value of the second objective at the interpolated solution. In



(a) Adjacent vertices for interpolation



(b) Training setup for the advanced network

Figure 4.7: A spherical sensor with multiple objectives. (a) The original solution for objective *interest* (green) is marked as yellow, while the interpolated solution, which intersects the high-resolution solution (gray), is marked with a golden sphere. The interpolated solution is inside the triangle, which is spanned by the original solution and the adjacent vertices, marked as blue spheres. (b) The training setup of the advanced network. The green sphere (interest) moves in the same way as in the basic setup. The red sphere (danger) moves parallel to the hull of the spherical sensor, around the green sphere. Additionally, the red sphere translates toward the agent.

different setups we use multiplication, concatenation, or addition as merge layer. The advanced network structure is shown in Figure 4.8. The accuracy of the first objective is similar to the multiple-maxima network since the topology is the same. The output of the second objective achieves a poor accuracy of 7.85% and 0.4 loss on the test data. Note that this are only theoretical values, and an evaluation on test scenes in Unity is more expressive.

Training Setup

As we have multiple inputs and outputs, we need a new training setup that considers both objectives. The possible values of the second objective need to be combined with the possible values of the first one. Therefore, we use an additional *danger* target that rotates in a plane around the *interest* target, which is oriented towards the agent, such that it is orthogonal to the sphere's surface. Thus, the rotation of the *danger* target is parallel to the sphere's hull and tangent to the *interest* rotation. That way, the *danger* values vary constantly for a single *interest* value with an alternating radius. The *danger* target is translated toward the agent, while its rotation radius decreases linearly. Hence, the outer rotations have a larger radius and the inner rotations have a smaller one. Thus, the movement describes a cone to keep a constant angle to the receptors. When the *danger* target arrives at the agent, the *interest* object is translated toward the agent, end-position, respectively, by updating its radius r using Equation 4.1. As a result, we prevent bias by ensuring that each receptor receives the same values from a rotation of the *danger* target in combination with the *interest* target. We get ≈ 570000 training samples in total. The new training setup is shown in Figure 4.7b.

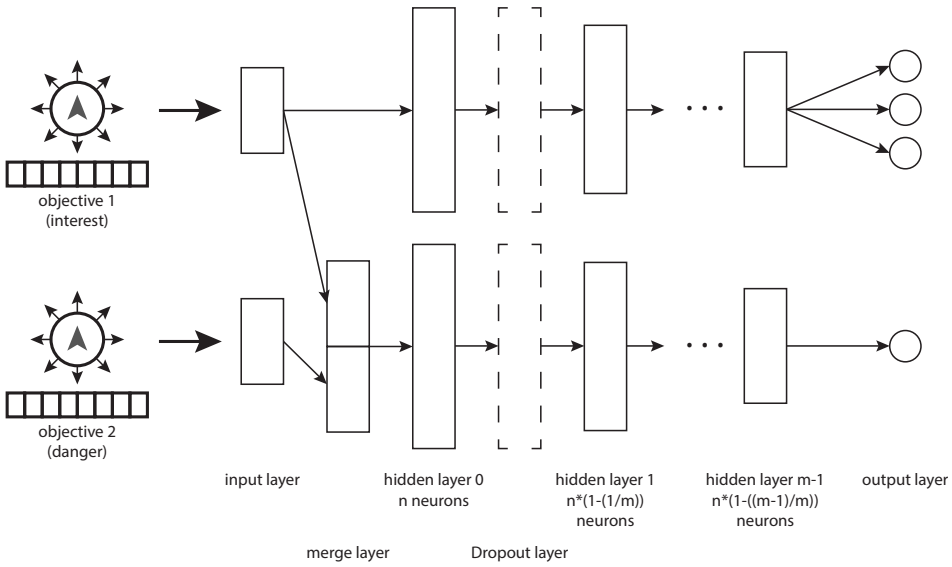


Figure 4.8: The topology of the advanced network. There are two inputs, one for each objective. The upper network is the same as in the basic configuration for multiple maxima. The lower network is similar but has an additional merge layer that combines both inputs. The output of the upper part is a vector in $\mathbb{R}^3$. The output of the lower part is a scalar in $\mathbb{R}$. All layers are fully connected.

5. Experiments and Evaluation

In this chapter, we benchmark our methods to compare its performance to original solutions based on a high-resolution sensor as state of the art. We first talk about the test setups and explain the metrics that we use. Afterward, we take a detailed look at the evaluation results of the network output, the computation times, and the network parameters. Finally, we discuss some notable remarks that we have observed.

5.1 Test Setups

5.1.1 Laboratory Scenes

Our test setups cover different laboratory scenes of different complexity. We split our test scenes into single objective, multi objective, and within these splits into single and multiple target objects. The before mentioned scenes use static agents to ensure that the measurements are unbiased with regard to the interpolated solution and the interpolated objective values. In two more complex scenes, we measure the agent's ability of avoidance and interpolation in movement. An overview of all test setups is shown in Figure 5.1

Single Objective And Single Target

This setup consists of a single target of objective *interest* that rotates with a dynamic radius around the fixed agent. The setup is likewise the training setup from Section 4.2.3. Its purpose is to show the theoretically best interpolation of a network.

Single Objective And Multiple Targets

As we stated in Section 4.3, multiple maxima have a significant influence on the interpolation process. Thus, this setup is similar to the setup above but consists of multiple targets that rotate around the fixed agent from different directions at different speeds in parallel. Here we compare the performance of the networks for the multiple maxima problem.

Multiple Objectives And Single Targets

In addition to the setup with a single target of a single objective, two additional targets of the second objective *danger* are added to the scene. The setup is comparable to Section 4.4.2 except that we now use two *danger* targets that rotate perpendicular to each other in different frequencies, so that the second objective has values in each of the three dimensions around the first objective. That way, we test the theoretically best interpolation of both objectives.

Multiple Objectives And Multiple Targets

This is an expansion of the multiple maxima setup for the second objective. Thus, similar to the setup before, the *danger* targets rotate perpendicular around two of the *interest* targets. That way, we can compare the performance with multiple objectives and multiple targets as kind of a hard test scenario.

Complex Path Avoidance Scene

The last scene shows a city and an airplane agent, such that it is close to a real simulation. It consists of an *interest* path that the agent should follow. Additionally, the boundary of nearby buildings and spheres serve as *danger* obstacles, which should be avoided. The scene consists of a high amount of target objects, and thus, it causes a high computational effort. Hence, this scene is a benchmark for the computational performance of the system.

5.1.2 Metrics

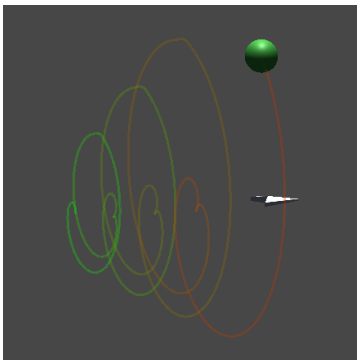
To compare the results of the different agents, sensors, respectively, we need to find metrics that describe the demands on the sensors and distinguish them. As ground truth, we use the high-resolution sensor. Therefore, we define the following metrics for each time step:

Angle deviation defines the deviation of the decided direction in comparison to the output of the ground truth in degrees.

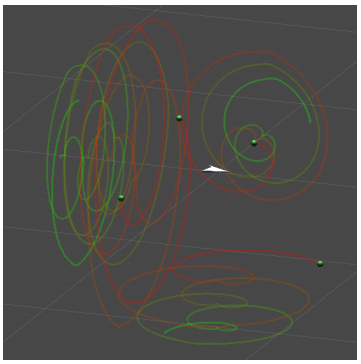
Angle constancy defines the maximum deviation between the current decided direction in time step t and the last $t - 5$ time steps in degrees.

Objective 1 deviation defines the deviation of the first objective value obtained from the magnitude of the decision vector in comparison to the ground truth.

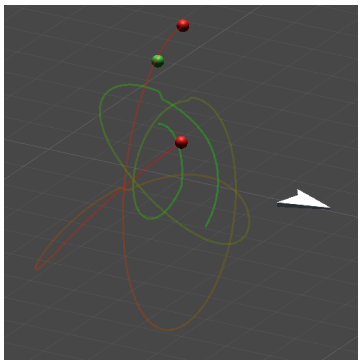
Interpolated objective 1 deviation defines the deviation of the first objective value obtained from linear interpolation in comparison to the ground truth. This is only available for the solution of the neural network.



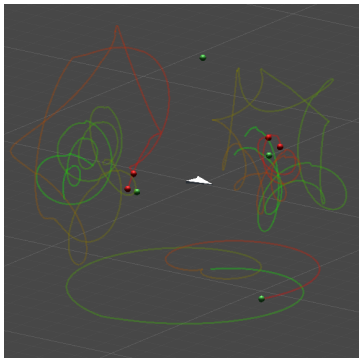
(a) Single objective and single target



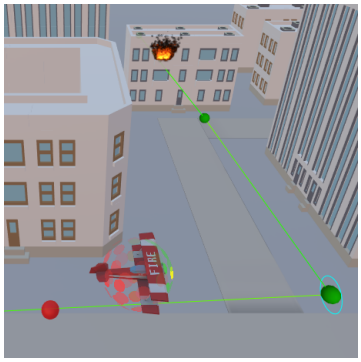
(b) Single objective and multiple targets



(c) Multiple objectives and single targets



(d) Multiple objectives and multiple targets



(e) Complex path avoidance scene

Figure 5.1: Overview of the evaluation setups. The trajectories are colored from green to red with regard to their translation time. In (a) and (b), the trajectories of the *interest* targets are shown. (c) shows the trajectories of the *danger* targets, while (d) additionally marks the trajectory of the interest target below the agent. (e) shows an excerpt of the path scene, where the agent is marked as an airplane. The *danger* targets are many buildings and red spheres. The path marks the consecutive green *interest* targets.

Objective 2 deviation defines the deviation of the second objective value obtained from the magnitude of the decision vector in comparison to the ground truth.

Interpolated objective 2 deviation defines the deviation of the second objective value obtained from linear interpolation in comparison to the ground truth. This is only available for the solution of the neural network.

5.2 Evaluation Results

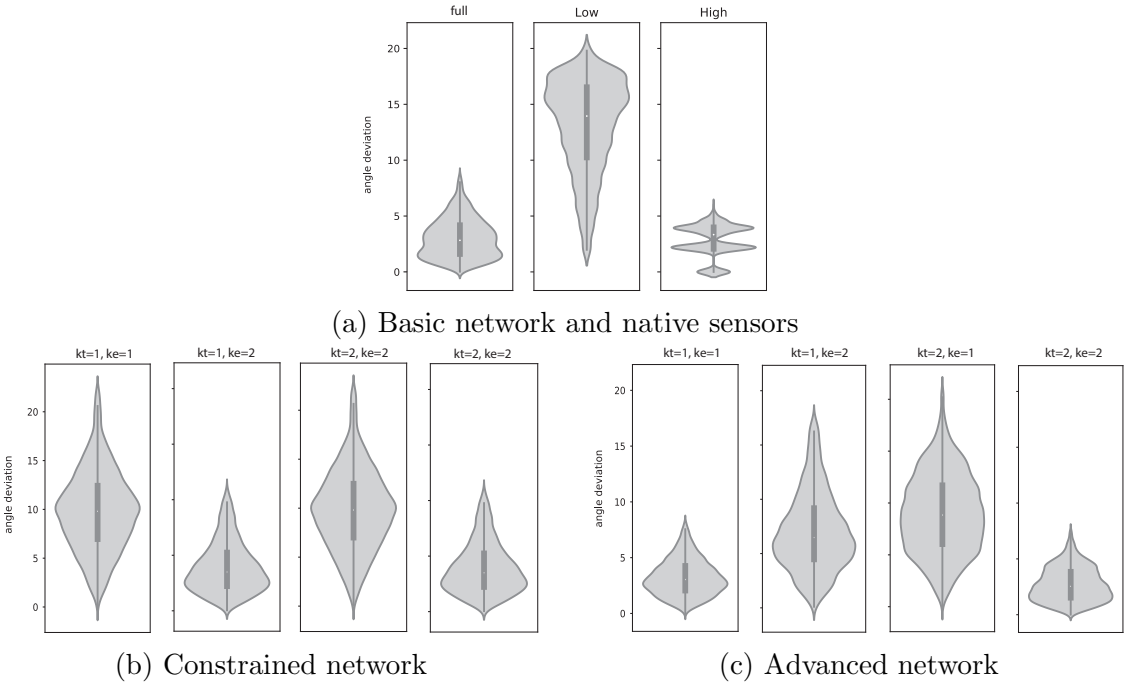
In this section, we discuss the results of the evaluation process. Based on the results, we give recommendations for our network topology and neighborhood structure. We compare the performance of state of the art to our method and show the limitations. Note that all tables that we refer to are in the appendix. Note that we filter outliers in our violin plots using the Interquartile Range (IQR). Therefore, the lower boundary is $1.5 \cdot Q_1$, and the upper boundary is $1.5 \cdot Q_3$. The results in the tables contain all data, including the outliers. The violin plots show statistical values as well as the data distribution.

5.2.1 Network Accuracy

We compare our method to three different sensor resolutions: low-resolution (42 receptors), medium high-resolution (642 receptors), and very high-resolution (10242 receptors). The ladder is our target sensor from the preceding chapters. It serves as ground truth so that all others are compared to it by their deviation to its values. For each of our approaches, we compare different configurations. Note that we first compare all configurations and focus on the best ones in the subsequent tests.

Single First Objective

As one can see in Table A.1, we are able to achieve better results than the medium-high sensor with several setups. As the median angle deviation shows, the results of our best configurations are in the range of 2.6 - 3.7, compared to 3.3 of the medium-high sensor. Additionally, the angle constancy shows, that the best configurations are less constant, resulting in a smoother movement with fewer jumps between the solutions similar to the ground truth. Note that a sensor with a low angle constancy has fewer changes between the solutions, and thus, remains at the same receptor, even if the target is moving. Additionally, this is shown in Figure 5.2, where smooth solutions result in smaller quantiles, are more compact, and have fewer extrema. To find the optimal configuration, we train and test our network with different constrained k-neighborhoods, as introduced in Section 4.3. We denote the constraints as kt for the k-neighborhood in the training process and ke for input k-neighborhood in the evaluation. While the unconstrained network, referred to as *full* in the figures and tables, yields the best results without any constraint ke, it suffers from constraints resulting in values close to the low-resolution input sensor. Interestingly, both constrained sensors in Figure 5.2b yield better results if their input ke equals 2, regardless of their training neighborhood kt. In contrast, the advanced network performs better if kt and kt are equal, as shown in Figure 5.2c. Even though the median magnitude deviation for objective 1 is small compared to the low-resolution sensor and the medium high-resolution sensor, but as Figure 5.3a and 5.3b shows, the deviation has a wider range. The best result is achieved by the advanced network with $kt = k2 = 2$. The magnitude deviation is significantly smaller with a narrow range, as shown in Figure 5.3c.



(a) Basic network and native sensors

(b) Constrained network

(c) Advanced network

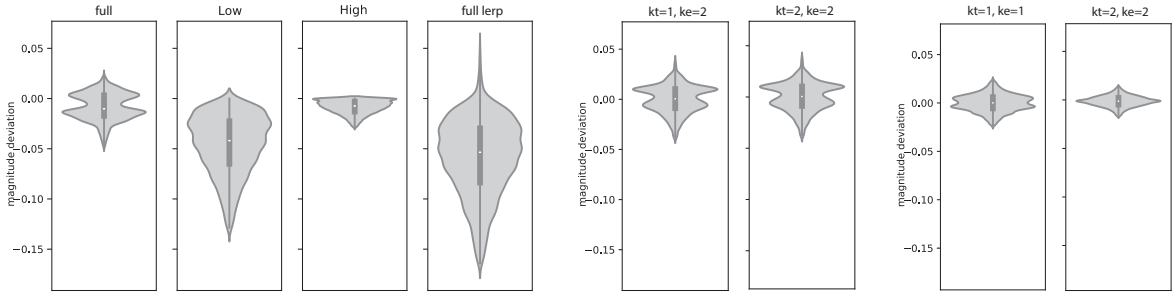
Figure 5.2: Angle deviation of the different sensors and networks in evaluation scene 1. (a) shows the basic network (full) and the native sensors. (b) and (c) show the influence of different k-neighborhoods in the training (kt) and in the evaluation (ke) for the constrained network, and the advanced network.

Multiple First Objective

In this setup, we can see the failure of the unconstrained sensor with multiple maxima. Table A.2 shows that all networks suffer from the additional targets, but can compensate due to the constrained k-neighborhood. The medium high-resolution sensor can show his strength in this setup with a nearly unchanged angle deviation and a slightly worse magnitude deviation as shown in Figure 5.4a and Figure 5.5a. The advanced network configuration with $kt = ke = 1$ achieves the best results of our networks. Even though the median angle deviation of the advanced network is small, the range of the extreme values is much larger, as one can see in Figure 5.4c. Note that due to the different sensor resolutions, the computed direction of the very high-resolution ground truth sensor (and the medium high-resolution sensor) may strongly differ from the low-resolution sensor, that serves as the input for our network. This phenomena is explained in Section 5.2.4 in more detail. Thus, huge differences may not lead to wrong computations but affect the statistics, especially for extreme values. Though the angle deviation is worse than the medium high-resolution sensor, the magnitude deviation is better and has a smaller variance, as one can see in Figure 5.5c.

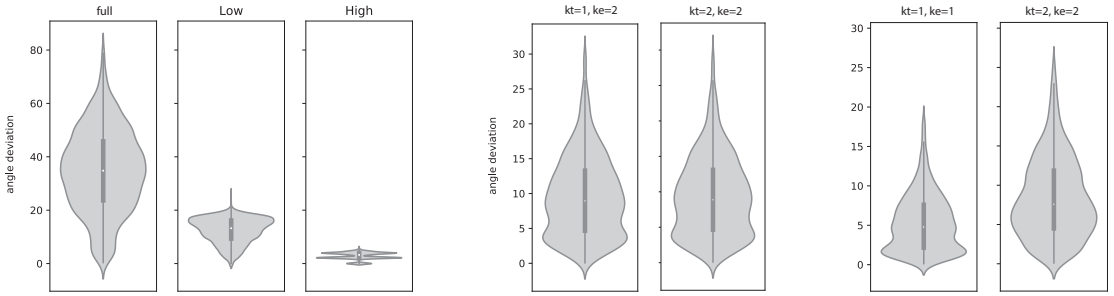
Single First And Second Objective

This setup focuses on the estimation of the second objective. Therefore, we compare the computed results of the original sensors to the linearly interpolated values of the basic



(a) Basic network and native sensors (b) Constrained network (c) Advanced network

Figure 5.3: Primary objective value deviation of the different sensors and networks in evaluation scene 1. (a) shows the basic network (full) and the native sensors. (b) and (c) show the similar results of the k-neighborhoods in the training (kt) and in the evaluation (ke) for both, the constrained network and the advanced network.



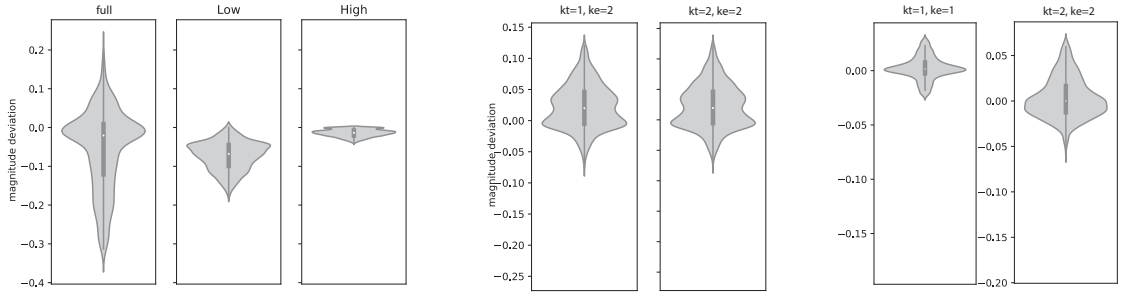
(a) Basic network and native sensors (b) Constrained network (c) Advanced network

Figure 5.4: Angle deviation of the different sensors and networks in evaluation scene 2. (a) shows the basic network (full) and the native sensors. (b) and (c) show the results of the k-neighborhoods in the training (kt) and in the evaluation (ke) for the constrained network and the advanced network.

and constrained networks, and the computed values of the advanced networks. While the median of the low-resolution sensor in Table A.3 shows only a small deviation, the appurtenant distribution in Figure 5.6a reveals a larger range of the data, similar to the linearly interpolated solution of the basic network. Our results show that the estimation of our advanced network is poor compared to the results of the native sensors in Figure 5.6a or the linearly interpolated solutions in Figure 5.6b. The concatenated merge layer shows the best results for the advanced network, but their overall performance is not as good as expected. Even though the majority of the advanced network results have a small deviation, their range of results is rather large as Figure 5.6c and 5.6d shows.

Multiple First And Second Objective

Similar to the results of setup 2, the performance of all sensors is decreased due to the multiple targets. As Table A.4 indicates, the medium-high sensor benefits from its



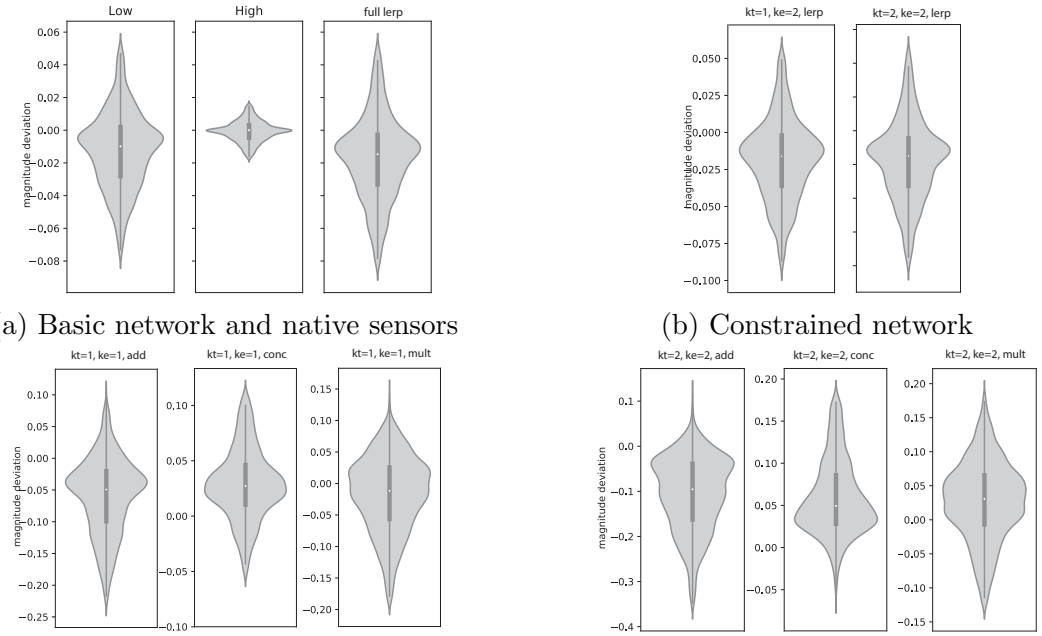
(a) Basic network and native sensors (b) Constrained network (c) Advanced network

Figure 5.5: Primary objective value deviation of the different sensors and networks in evaluation scene 2. (a) shows the basic network (full) and the native sensors. (b) shows similar results for both k-neighborhoods for the constrained network, while (c) shows differences for the $k-neigborhood$ in training (kt) and evaluation (ke) for the advanced network.

high resolution. Since the performance of the advanced network is poor even in the less complex setup 3, the performance is worse in this setup as Figure 5.7c and 5.7d points out. Again, it is best to estimate the second objective by using linear interpolation, as one can see in Figure 5.7b.

5.2.2 Computation Times

We measure the computation times on a machine running Windows 10 on an AMD Ryzen 7 1700 with a total number of 8 native cores on 3.7GHz speed and 3000MHz RAM. Our network needs about 0.66ms using the basic network topology and about 0.79ms with the advanced network topology to compute the results, including the preprocessing of the neighborhood. Note that we do not take the overhead of the Tensorflow backend that is running in the background into account. The computation time of our system is independent of the specific scene setup, and thus, the performance relies on the sensor resolution only. As we can see in Table 5.1, the sensor resolution has a huge impact on the computation time. While sensors with a higher resolution are quite fast with a low number of targets, they become very slow with a higher number of targets. Especially in real-world simulations, where many environmental influences must be accounted for, the number of targets can increase very fast. Thus, only agents with a low-resolution sensor are able to perform in real-time. Especially in the complex evaluation scene, the number of targets is very high. If we use so-called *bounds* behaviors that perform raycasts to perceive the shape of an object, the computational costs explode for the high-resolution sensors so that the scene stuck. Again, the computation time of our network is not affected by the number of targets at all. Thus, we can achieve a high-quality resolution at a very low constant cost. Note that the times have been measured in debug mode. Thus the build mode is faster, but the problem of scaling remains.



(a) Basic network and native sensors

(b) Constrained network

(c) Advanced network $k = 1$

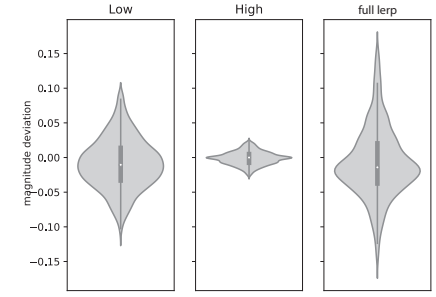
(d) Advanced network $k = 2$

Figure 5.6: Secondary objective value deviation of the different sensors and networks in evaluation scene 3. The values marked as lerp are interpolated since the networks do not have a direct output for the second objective. (a) shows the basic network (full) and the native sensors. (b) shows for both of the constrained networks. (c) and (d) shows the influence of different types of merge layers on the estimation of the second objective.

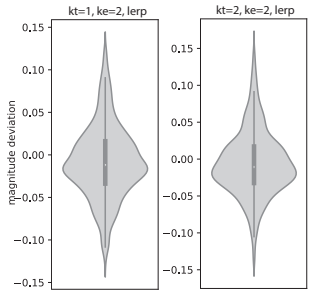
5.2.3 Network Topology

Training Data

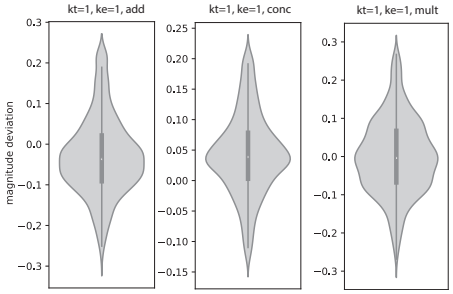
The training data has a major influence on the performance of the network. If we use too many of the same data, the network overfits. Furthermore, the network overfits if we train too many epochs. As we can see in Figure 5.2b and 5.2c, the performance of the network differs. Though the topology is the same for both the constrained and the advanced network, the latter performs better. Additionally, there is a major difference between their performance for different ke. While the constrained network works best with $ke = 2$ for $kt = 1$ and $kt = 2$, the advanced network works best with $kt = ke$. All of them perform significantly worse if these conditions are not fulfilled. We assume that this is a consequence of the significantly higher amount of training data for the advanced network. As mentioned in Section 4.4.2, the training focus is on the variance of the second objective in combination with the first objective. Hence, there are far more samples of the same data for the first objective. Thus, the advanced network possibly overfits, but our results show that the performance is better, even in new circumstances like the multiple target scene. We assume that the constrained network achieves a



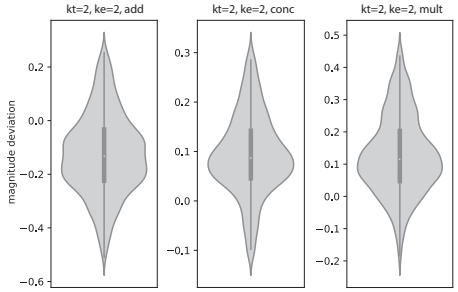
(a) Basic network and native sensors



(b) Constrained network



(c) Advanced network $k = 1$



(d) Advanced network $k = 2$

Figure 5.7: Secondary objective value deviation of the different sensors and networks in evaluation scene 4. The values marked as lerp are interpolated since the networks do not have a direct output for the second objective. (a) shows the basic network (full) and the native sensors. (b) shows for both of the constrained networks. (c) and (d) show the influence of different types of merge layers on the estimation of the second objective.

similar performance and behavior with regard to kt and ke, with more training data or training epochs.

Network Topology

The network layers have a central role in the network's performance. The number of neurons determines the network's ability to fit the data. While we found the setup of the basic network, as mentioned in Section 4.3, to be optimal, the performance of the advanced network is poor. Especially, the prediction of the second objective is poor. We increased the number of neurons in the hidden layers. In Figure 5.8, we can see that twice as many neurons have only a minor impact on the performance. In contrast, the type of merge layer that connects the input of both context maps, objectives, respectively, has an impact on the performance. As we show in Figure 5.6c and 5.6d, a concatenation of both inputs performs best.

5.2.4 Notable Remarks

Interestingly, a training on $kt = 1$ yields more reliable results than $kt = 2$ in terms of multiple maxima but tends to perform small jumps if the decided receptor switches. If

Table 5.1: Evaluation of the computation times. The time is measured in ms for different sensor resolutions in different scene setups.

Scene setup	Low	Medium High	Very High
1 target	0.12	0.99	15.1
100 targets	2.13	27.52	420.1
Path scene with basic behavior	9	44	551
Path scene with raycast behavior	14.5	144	2471

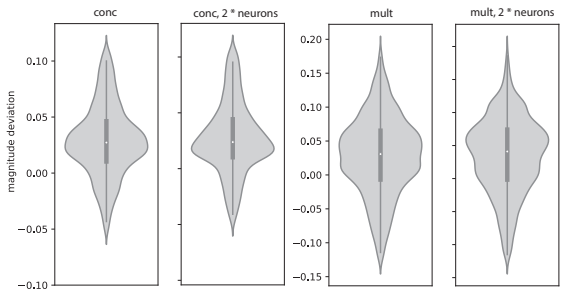


Figure 5.8: The influence of the number of neurons in the hidden layers for different types of the advanced network.

we increase ke in the simulation but keep the trained network with $kz = 1$, the results become more stable with respect to the small jumps. However, the process is more sensitive to additional targets, and thus, resulting in falsified interpolation. In contrast, a training neighborhood of $kt = 2$ is more accurate, but more sensitive to additional targets as well. A pre-trained network with $kt = 2$, and $ke = 1$ in the simulation produces hard bounces since the interpolation stays close to the original solution.
Furthermore, we observe huge differences between the performance of each of our networks between Table A.1 and Table A.3, and Table A.2 and Table A.4 for the first objective, even though the scene setup is the same for this objectives. We assume, that the additional targets cause such high computational costs for the high-resolution sensors, that the update cycle is affected, and thus, the values differ in time. Since the very high-resolution sensor consists of 10242 receptors, and scene 2 and 4 consist of 4 or rather 8 targets, a maximum of 81936 dot products are computed each update cycle, and the resulting MCO problem must be solved.
Remarkable is the effect of the sensor resolution on the actual decision. They may differ entirely in the case of multiple targets, as one can see in Figure 5.9. This effect occurs due to the higher perception ability of the high-resolution sensor since it can perceive values that are high in a very small domain on the sensor. Thus, the low-resolution sensor may have higher values for another target, since its receptors do not point as

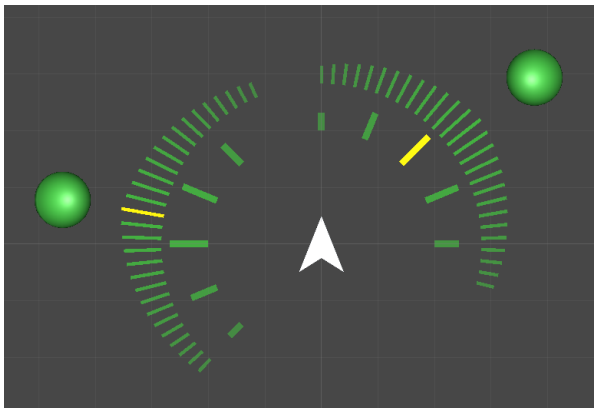


Figure 5.9: The anomaly between the sensor decisions. The low-resolution sensor points toward the right target, while the high-resolution sensor points to the left target.

directly to the target as the high-resolution sensor does. This being said, the deviation of the interpolated solution of our network is only as good as the low-resolution sensor is correct. Thus, it can only find a local best optimum. Furthermore, large differences to the result of the high-resolution sensor may be no error with respect to the low-resolution sensor.
Last, we observed during runtime that the results of the advanced network for the second objective are better if the targets of both objectives are spatially close together. In this case, the estimated results of the advanced network are better than the interpolated values.

6. Conclusion and Future Work

This work successfully introduced a novel method to interpolate functions on spherical sensors. We applied our method to the field of autonomous steering agents that found their decisions on multi-criteria optimization. We combined the ideas of Super-Resolution, Supersamling, and machine learning. As input, we used the data of a low-resolution sensor. A deep neural network performs both tasks of Super-Resolution by upscaling the data, and Supersampling by fitting a function on the data to find the interpolated value by regression. Thus, we were able to find the optimum around a given receptor, data point, respectively. To overcome the problem of averaging, that arises from multimodal distributions of the sensor data, we used a constraint. Therefore, we limited the input data to a k-neighborhood around the original solution. We tested the influence of $k = 1$ and $k = 2$ during the training and the tests. As we showed in Section 5.2, both variants can produce accurate results. Furthermore, we introduced an advanced layout of our network that is able to predict the second objective of a multi-criteria problem. Unfortunately, the evaluation showed poor results for the prediction of the second objective. Thus, linear interpolation of the second objective was more accurate and reliable, though it is only a rough estimation. In contrast, the ability of the advanced network to predict the first objective, was even better than the constrained network, though the network topology is the same. We lead this back to the training process, where more data were presented to the advanced network, even if they were not diverse. In scenarios with a high count of targets, the high-resolution sensor showed benefits in terms of accuracy, but at the flaws of higher computation times and discontinuous solutions. Especially in scenarios with a massive amount of targets, the computation time increases so much, that it is not practical, as we showed in Section 5.2.2. In typical scenarios, there are less target divergent targets of the first objective, but more of the second one. Hence, our method is superior to the native sensor since the accuracy on the first objective is high, the computation times are low and constant, and thus, it is independent of the scene complexity, count of targets, respectively.

Future Work

The hyperparameters of the network can be further improved by an extensive grid search to obtain the best possible performance. To improve our method further for the multiple maxima problem, one could expand the training setup by additional targets. This way, a constrained neighborhood is learned in an environment with multiple targets, which possibly results in more accurate solutions with less variance. In addition, a better architecture must be provided for the advanced network since the performance is poor on the second objective. Instead of neural networks, RBF networks can be used. This way, the neighborhood could be expanded, or is not needed anymore since RBFs take the distance of the data into account. Furthermore, a system can be designed that does not only estimate the best solution and the value of the second objective. Instead of falling back to the original solution in case the constraint gets violated, the best solution with respect to the second objective could be found. Currently, our system is limited to steer a single agent by design as we provide only a single UDP connection and a single backend for prediction. By optimizing the communication via UDP, multiple agents could share the same connection. Thus, the input data from Unity must be serialized to transmit them as a single package to the Tensorflow backend. Alternatively, the network can be separated from Tensorflow and Python since it basically can be reduced to matrix multiplications, and thus, can be processed independent and directly in the sensor system. Hence, an instance for each agent can compute the solution, and thus, would be parallelizable on multiple threads.

A. Appendix

Table A.1: Results of evaluation scene 1: *single objective and single target.*

Agent Type	Angle		Deviation Objective 1	
	Constancy	Deviation	computed	lerp
MCO Agent				
low res	1.0000	13.9502	-0.0426	
high res	1.2000	3.3366	-0.0075	
very high res	3.8537			
Simple DNN, ke= evaluation neighborhood				
full sensor	3.5660	3.0402	-0.0104	-0.0529
full sensor, ke=2	3.1023	3.9202	-0.0143	-0.0490
full sensor, ke=1	1.1084	9.1345	-0.0973	-0.0400
Constraint DNN, kt = training neighborhood, ke= evaluation neighborhood				
kt=2, ke=full	3.4305	3.7278	0.0051	-0.0552
kt=2, ke=2	3.4616	3.6898	0.0000	-0.0528
kt=2, ke=1	1.3676	10.0766	-0.0708	-0.0396
Constraint DNN, kt = training neighborhood, ke= evaluation neighborhood				
kt=1, ke=full	3.4036	3.7456	0.0050	-0.0547
kt=1, ke=2	3.5086	3.7106	-0.0001	-0.0531
kt=1, ke=1	1.3702	10.0539	-0.0710	-0.0396
Advanced DNN, kt = training neighborhood, ke= evaluation neighborhood				
kt=2, ke=2	3.4985	2.6062	-0.0015	-0.0532
kt=2, ke=1	1.1003	9.0063	-0.0173	-0.0417
kt=1, ke=2	2.8926	6.6784	-0.0185	-0.0608
kt=1, ke=1	3.4966	3.1599	0.0000	-0.0521

Table A.2: Results of evaluation scene 2: *single objective and multiple targets.*

Agent Type	Angle		Deviation Objective 1	
	Constancy	Deviation	computed	lerp
MCO Agent				
low res	1.0000	13.8181	-0.0676	
high res	8.3937	3.3679	-0.0141	
very high res	9.0000			
Simple DNN, ke= evaluation neighborhood				
full sensor	6.0247	36.6016	-0.0222	-0.0239
full sensor, ke=2	5.9833	9.9775	0.0122	-0.0776
full sensor, ke=1	2.8313	10.7043	-0.1603	-0.0606
Constraint DNN, kt = training neighborhood, ke= evaluation neighborhood				
kt=2, ke=2	7.4422	9.7266	0.0200	-0.0881
kt=2, ke=1	3.3397	11.3354	-0.1366	-0.0644
Constraint DNN, kt = training neighborhood, ke= evaluation neighborhood				
kt=1, ke=2	7.3990	9.7604	0.0204	-0.0839
kt=1, ke=1	3.4043	11.6225	-0.1343	-0.0635
Advanced DNN, kt = training neighborhood, ke= evaluation neighborhood				
kt=2, ke=2	7.5536	8.1997	0.0004	-0.0935
kt=2, ke=1	2.9931	10.9340	-0.0399	-0.0633
kt=1, ke=2	6.8966	9.5843	-0.0307	0.0999
kt=1, ke=1	8.1297	5.4585	-0.0017	0.0829

Table A.3: Results of evaluation scene 3: *multiple objectives and single target*. Advanced networks with an additional 2 after the type of the merge layer, have a doubled amount of neurons in the hidden layers of the second part of the network for the additional objective. Merge layers are denoted as *mult* = multiplication, *conc* = concatenation, and *add* = addition.

Agent Type	Angle		Deviation Objective 1		Deviation Objective 2	
	Constancy	Deviation	computed	lerp	computed	lerp
MCO Agent						
low res	1.0000	13.9503	-0.0411		-0.0097	
high res	8.2061	3.2517	-0.0074		-0.0007	
very high res	5.2593					
Simple DNN, ke= evaluation neighborhood						
full sensor	4.7910	3.6053	-0.0102	-0.0533		-0.0136
Constraint DNN, kt = training neighborhood, ke= evaluation neighborhood						
kt=2, ke=2	4.6232	4.2604	-0.0002	-0.0526		-0.0141
Constraint DNN, kt = training neighborhood, ke= evaluation neighborhood						
kt=1, ke=2	3.9866	6.6393	0.0324	-0.0557		-0.0150
Advanced DNN, kt = training neighborhood, ke= evaluation neighborhood, type of merge layer						
kt=2, ke=2, mult	4.7619	3.4138	0.0014	-0.0527	0.0316	-0.0145
kt=2, ke=2, mult2	4.8388	3.2393	0.0014	-0.0518	0.0409	-0.0136
kt=2, ke=2, conc	4.6770	2.9787	-0.0018	-0.0524	0.0508	-0.0132
kt=2, ke=2, add	4.8236	3.1851	-0.0014	-0.0534	-0.0966	-0.0145
kt=1, ke=1, mult	4.7423	3.9390	0.0007	-0.0519	-0.0131	-0.0111
kt=1, ke=1, conc	4.8914	3.7587	0.0008	-0.0524	0.0280	-0.0133
kt=1, ke=1, conc2	4.6149	4.0070	0.0012	-0.0519	0.0247	-0.0136
kt=1, ke=1, add	4.6170	3.9344	0.0003	-0.0524	-0.0499	-0.0134

Table A.4: Results of evaluation scene 4: *multiple objectives and multiple targets*. Advanced networks with an additional 2 after the type of the merge layer, have a doubled amount of neurons in the hidden layers of the second part of the network for the additional objective. Merge layers are denoted as *mult* = multiplication, *conc* = concatenation, and *add* = addition.

Agent Type	Angle		Deviation Objective 1		Deviation Objective 2	
	Constancy	Deviation	computed	lerp	computed	lerp
MCO Agent						
low res	31.7174	13.8183	-0.0679		-0.0104	
high res	15.2619	3.3366	-0.0139		-0.0059	
very high res	13.3285					
Constraint DNN, kt = training neighborhood, ke= evaluation neighborhood						
kt=2, ke=2	12.9829	11.0337	0.0201	-0.0843		-0.0031
Constraint DNN, kt = training neighborhood, ke= evaluation neighborhood						
kt=1, ke=2	11.4318	8.8834	0.0881	-0.0894		-0.0073
Advanced DNN, kt = training neighborhood, ke= evaluation neighborhood						
kt=2, ke=2, mult	13.8365	10.5448	0.0008	-0.0863	0.1164	-0.0092
kt=2, ke=2, mult2	13.1998	10.4903	0.0018	-0.0884	0.1138	-0.0081
kt=2, ke=2, conc	12.8833	10.7572	-0.0009	-0.0886	0.0866	-0.0089
kt=2, ke=2, add	13.2440	10.9562	0.0002	-0.0913	-0.1307	-0.0100
kt=1, ke=1, mult	12.9148	8.2958	0.0011	-0.0895	-0.0026	-0.0080
kt=1, ke=1, conc	13.5491	8.0774	0.0013	-0.0839	0.0383	-0.0053
kt=1, ke=1, conc2	13.8149	8.6069	0.0026	-0.0839	0.0376	-0.0054
kt=1, ke=1, add	14.1056	8.6838	0.0028	-0.0839	-0.0321	-0.0057

Abbreviations and Notations

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
CAGD	Computer Aided Geometric Design
DNN	Deep Neural Network
ELU	Exponential Linear Unit
EM	Expectation Maximization
IQR	Interquartile Range
lerp	Linear Interpolation
MCO	Multi-Criteria Optimization
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
Slerp	Spherical Linear Interpolation
SVD	Singular Value Decomposition

List of Figures

List of Tables

Bibliography

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283. Version 1.10.0. (cited on Page 20 and 27)

[Buhmann, 2003] Buhmann, M. D. (2003). *Radial Basis Functions*. Cambridge University Press. (cited on Page 3 and 17)

[Burger and Burge, 2008] Burger, W. and Burge, M. J. (2008). *Digital Image Processing: An Algorithmic Introduction using Java*. Springer. (cited on Page 12)

[Chollet et al., 2018] Chollet, F. et al. (2018). Keras. https://keras.io. Version 2.2.2. (cited on Page 20)

[Clevert et al., 2015] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *Under Review of ICLR2016 (1997)*. (cited on Page 22)

[Crow, 1981] Crow (1981). A Comparison of Antialiasing Techniques. *IEEE Computer Graphics and Applications*, 1(1):40–48. (cited on Page 4 and 20)

[Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22. (cited on Page 19)

[Du et al., 2018] Du, P., Zhang, J., and Long, J. (2018). Super-Sampling by Learning-Based Super-Resolution. In *Algorithms and Architectures for Parallel Processing*, pages 76–83. Springer International Publishing. (cited on Page 4)

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159. (cited on Page 23)

[Farin and Hansford, 2000] Farin, G. E. and Hansford, D. (2000). *The Essentials of CAGD*. A. K. Peters, Ltd., Natick, MA, USA, 1st edition. (cited on Page 13 and 14)

[Fray, 2013] Fray, A. (2013). Steering Behaviours Are Doing It Wrong. https://andrewfray.wordpress.com/2013/02/20/steering-behaviours-are-doing-it-wrong/. Accessed: 2019-10-11. (cited on Page 7)

[Fray, 2015] Fray, A. (2015). Context steering: Behavior-driven steering at the macro scale. In Rabin, S., editor, *Game AI Pro 2*, pages 183–193. CRC Press. (cited on Page 1, 3, 7, and 14)

[Glorot et al., 2011] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. PMLR. (cited on Page 22)

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. Adaptive computation and machine learning. MIT Press. (cited on Page 20)

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*. Springer. (cited on Page 23)

[Kim et al., 2016] Kim, J., Kwon Lee, J., and Mu Lee, K. (2016). Accurate image super-resolution using very deep convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (cited on Page 4 and 20)

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*. (cited on Page 23)

[Kirst, 2015] Kirst, M. (2015). Multi-criteria optimized context steering for autonomous movement in games. Master's thesis, Otto-von-Guericke Universität, Magdeburg. (cited on Page 1, 3, 7, 14, and 31)

[Kruse et al., 2016] Kruse, R., Borgelt, C., Braune, C., Mostaghim, S., and Steinbrecher, M. (2016). *Computational Intelligence: A Methodological Introduction*. Springer-Verlag GmbH. (cited on Page 20)

[Ledig et al., 2017] Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., and Shi, W. (2017). Photo-realistic single image super-resolution using a generative adversarial network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (cited on Page 4 and 20)

[McDonald et al., 2007] McDonald, D. B., Grantham, W. J., Tabor, W. L., and Murphy, M. J. (2007). Global and local optimization using radial basis function response surface models. *Applied Mathematical Modelling*, 31(10):2095–2110. (cited on Page 4)

[Miettinen, 1998] Miettinen, K. (1998). *Nonlinear Multiobjective Optimization*. Springer US. (cited on Page 8)

[Polarith UG, 2019] Polarith UG (2019). Polarith AI. https://polarith.com/ai/. Version 1.7, Accessed: 2019-09-25. (cited on Page 27)

[Popko, 2012] Popko, E. S. (2012). *Divided Spheres*. Taylor & Francis Inc. (cited on Page 11)

[Renka, 1984] Renka, R. J. (1984). Interpolation of data on the surface of a sphere. *ACM Transactions on Mathematical Software*, 10(4):417–436. (cited on Page 3)

[Reynolds, 1999] Reynolds, C. W. (1999). Steering behaviors for autonomous characters. In *Game Developers Conference*, pages 763–782, San Jose, California. Miller Freeman Game Group. (cited on Page 1, 3, 5, 6, and 7)

[Shirley et al., 2009] Shirley, P., Ashikhmin, M., and Marschner, S. (2009). *Fundamentals of Computer Graphics*. A K Peters/CRC Press. (cited on Page 11)

[Shoemake, 1985] Shoemake, K. (1985). Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques - SIGGRAPH85*. ACM Press. (cited on Page 13)

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958. (cited on Page 23)

[Trefethen and Bau, 1997] Trefethen, L. N. and Bau, D. (1997). *Numerical Linear Algebra*. SIAM. (cited on Page 3, 17, and 18)

[Unity Technologies, 2019] Unity Technologies (2019). Unity Game Engine. https://unity.com. Version 2019.1, Accessed: 2019-09-25. (cited on Page 27)

[Xu, 2004] Xu, Y. (2004). Polynomial Interpolation on the Unit Sphere and on the Unit Ball. *Advances in Computational Mathematics*, 20(1/3):247–260. (cited on Page 3)

I hereby declare that I have written the present work myself and did not use any sources or tools other than the ones indicated.

Magdeburg, November 06, 2019