A Learning Approach for Optimizing Robot Behavior Selection Algorithm

Basile Tousside, Janis Mohr, Marco Schmidt and Jörg Frochte

Bochum University of Applied Science, 42579 Heiligenhaus, Germany,
{basile.tousside, janis.mohr, marco.schmidt, joerg.frochte}@hs-bochum.de

Abstract. Algorithms are the heart of each robotics system. A specific class of algorithm embedded in robotics systems is the so-called behavior – or action – selection algorithm. These algorithms select an action a robot should take, when performing a specific task. The action selection is determined by the parameters of the algorithm. However, manually choosing a proper configuration within the high-dimensional parameter space of the behavior selection algorithm is a non-trivial and demanding task. In this paper, we show how this problem can be addressed with supervised learning techniques. Our method starts by learning the algorithm behavior from the parameter space according to environment features, then bootstrap itself into a more robust framework capable of self-adjusting robot parameters in real-time. We demonstrate our concept on a set of examples, including simulations and real world experiments.

Keywords: Robot learning · Behavior selection · Parameter optimization

1 Introduction

Modern robots are nowadays empowered with more and more behavior selection algorithms (BSA) [6, 15]. A behavior is an action a robot can take in response to objectives or sensor inputs. These behaviors might be either the algorithm of interest or a component of an algorithm with a larger scope. A behavior selection algorithm therefore selects appropriate behaviors or actions for a robot to perform. It is used in various applications to allow robots to autonomously perform diverse tasks like vision, navigation or grasping [8, 13].

Robot behavior selection algorithms often involve many parameters that have a significant impact on the algorithms efficiency. In this setting, a key challenge concerns the tuning of those parameters. Consider for example a parametric path planning algorithm operating in a dynamic environment. A parameter configuration that avoids a moving human is not guaranteed (without further tuning) to be effective at avoiding other dynamic obstacles like a random moving cat for example. The parameters tuning of behavior selection algorithms is sometimes delegated to a human and accomplished through a process of trial-and-error [1]. Unfortunately, manual tuning of robot parameters is not only time consuming, it also tends to make the algorithms vulnerable. As a result, the question of how

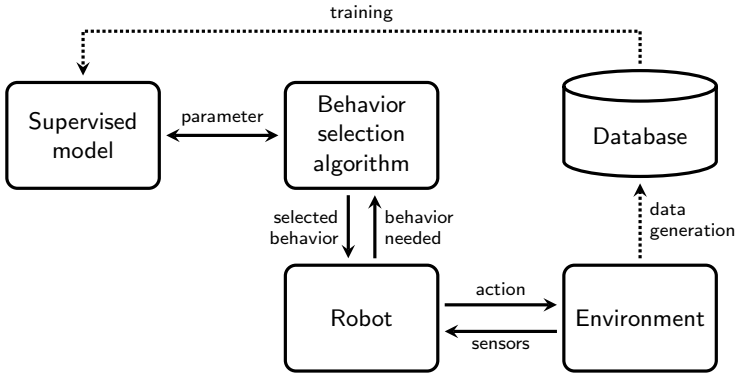

Fig. 1: Our strategy for implementing a learnable module, which gives a robot the ability to self-adjust its parameters while performing a specific task. The dashed lines represent connections taking place exclusively at the first stage of our method, which focus on learning the parameters of the BSA w.r.t environment features.

to optimally select parameters of robot behavior selection algorithms is still an open problem.

In this paper, we demonstrate that these parameters can be learned from observation data via supervised learning techniques. This automates the choice of parameters therefore providing an appealing alternative to the complex manual tuning. Another significant benefit is that the learning solution will approximate an optimum in contrast to manual solutions which are often not of the best quality. Our approach is to build a supervised learner, which learns the parameters of a behavior selection algorithm – therefore learning the behavior of the behavior selection algorithm – from observation data. This supervised learner is then later used to optimize the parameters of the behavior selection algorithm for the specifics of the environment. This is schematically illustrated in Figure 1. Other authors have already approached the problem this way. However, the learning framework they propose almost always uses specialized techniques such as salient landmarks [12], language measure [14] or stochastic Petri nets [10]. Our framework in contrast uses supervised learning technique making it general and transferable to any other robotics parameter optimization problem. It starts with a data generation process and bootstraps itself towards a more general framework capable of self-adjusting robot parameters while the robot is performing a given task. One challenge when using supervised learning technique is the need of labelled data. Our framework takes this into account by incorporating a data generation phase, which will be presented in Section 3.6.

In this paper, we focus on the path planning behavior selection problem, which is a crucial component of each robot navigation framework. When navigating in a dynamic environment, a path planning algorithm – which is a behavior selection algorithm – might be confronted with the issue of which behavior to adopt with respect to the environment, that is, in order to accurately avoid a

specific obstacle for example. The behavior the path planning algorithm adopts is determined by its parameters. The problem can then be rephrased as follows: Which parameters should be chosen to avoid a certain type of obstacle. Later in this paper, we will demonstrate how our method is able to solve this problem by learning from observation data the behavior of the path planning algorithm when avoiding obstacles.

One problem that can pose serious difficulties to the learning algorithm is that for some behavior selection algorithm like path planning, searching for the optimal parameters belongs to the class of the so-called ill-posed problems since there is not a unique solution. In fact, different paths produced by various configurations in the parameter space might not significantly differ in terms of quality. To overcome the ill-posedness of parameter search, the training data needs to be filtered and cleaned-up, which is done in a preprocessing step.

Another challenge when applying parameter learning to behavior selection algorithm is the fact that robotic algorithms are mostly interconnected. Consequently, integrating external factors such as learned parameters in one algorithm without compromising the functionality of some others is not trivial. This difficulty is well illustrated in our path planning use case. In fact, robot navigation can mostly be achieved by successfully connecting several key components like map building, localizers, path planner or range sensors among others. However, as the number of components increases, their relationships become difficult to manage. As a side effect, if we learn the behavior of a path planning algorithm, the problem arises of how to integrate the learned behavior into the navigation framework without comprising the functionality of the overall framework. To solve this problem, a framework, which formalizes the navigation system and its components was required. This will be revisited in Section 3.4. The main contributions of this paper are as follows:

- Strategy that generates labelled data for a robot behavior selection algorithm from environment observations.
- Supervised-learning approach for learning parameter (and thus the behavior) of robot behavior selection algorithms.
- Framework that exploits a supervised learned model to optimally adjusting robot parameter in real-time.

2 Methodology

We aim at demonstrating how to empower robots with the ability to automatically tweak their parameters, therefore reacting to environment changes. In our setting, the actions – or behaviors – of the robot are controlled by a behavior selection algorithm. Therefore, by controlling the behavior selection algorithm we can indirectly control the robot behavior (see Figure 1). Since behavior selection algorithms are parameter-dependent, they can be controlled via their parameters. Our method builds on this and can be subdivided into two steps, first, it focus on learning the parameters of the behavior selection algorithm with respect to some environment features. Doing so, it somehow learns the behavior

Algorithm 1: Our method for optimizing parameters of BSA

Input: Deterministic behavior selection algorithm $\mathcal{A}$ with parameters P e.g., parametric path planning algorithm (RRT, APF)

Result: Machine learning model, which improve in real-time the robot behavior with respect to a specific environment

1 parameterize the environment with some features $\mathcal{X}$;
2 generate observation data with features $\mathcal{X}$ and response P ;
3 train a supervised-learning model $\mathcal{M}$ to learn P w.r.t $\mathcal{X}$;
4 integrate $\mathcal{M}$ into the robot framework to which $\mathcal{A}$ belong e.g., navigation framework ;
5 robot use $\mathcal{M}$ to self-adjust its behavior by optimizing P according to environmental conditions ;

of the behavior selection algorithm according to environmental conditions. In a second step, the learned model is exploited to optimize in real-time the parameters of the behavior selection algorithms for the specifics of the environment. Let us illustrate this with a concrete use-case. In a navigation setup, path planning is a typical behavior selection algorithm. It selects the appropriate behaviors or actions for a robot to take in response to environment input (such as free space, obstacles, etc.) gained by robot sensors. Consider an environment including a static chair and two dynamic obstacles, namely, a deterministic moving human and a random moving cat. A robot equipped with a parametric path planner – such as Rapidly-exploring random tree (RRT) [11] or the artificial potential field (APF) [9] method – is required to navigate that environment while avoiding obstacles. Thus, when facing an obstacle, the robot – in fact the path planning algorithm – should decide which behavior (or action) to select. More concretely, the path planning algorithm, which is here the behavior selection algorithm should decide which parameter configuration to use. It is difficult to manually find near-optimal choice of parameter configuration to avoid all three types of obstacles. In fact, it can be expected that a skilled human will often be able to do an as good job as machine learning at finding parameters that avoid each obstacle individually, especially for the chair and the human. But for avoiding all the obstacles sequentially, machine learning will be of great help for novice as well as expert users.

Our method for using machine learning to solve such a behavior selection problem consists of first (i) learning the parameters of the path planning algorithm according to some environment feature such as the obstacles's behavior. To do so, we build a labelled data-set, where features are environment characteristics (obstacle type, obstacle speed, etc.) and responses are the parameters to be learned. This is carried out in a bootstrap process involving an intelligent grid-search and a compact local search. This is revisited in Section 3.6. Once the data are collected and preprocessed, regression techniques can be used to learn path planning parameters corresponding to environment features. However, as pointed out earlier, parameter learning is often an ill-posed problem, since different parameter configurations can lead to an equally good result. This

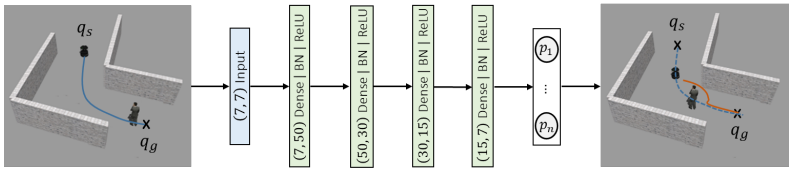

Fig. 2: Our learning model: Predicted parameters are fed to the robot via the control system. Input and output dimensions of each layer are shown on the connections.

is the case for our path planning parameter learning use-case as well as for many other robotics problems like the inverse kinematic [2]. To tackle such problems using standard regression technique like a multilayer Perceptron (MLP) or a k-nearest neighbors (KNN), the labelled data needs to be filtered and cleaned up as will be presented in the experiments section, otherwise those techniques will fail at finding good solutions as shown in [4]. Once the data is generated and preprocessed, they can be fed to an appropriate regression model, which learns to predict the parameter values the robotics algorithm should use in real-time.

The second step of our method for solving the path planning behavior selection problem is to (ii) deploy the learned model in the navigation framework, in order to optimize the path planning parameter - therefore controlling and improving the path planning behavior - regarding environment features. Our general framework for solving the BSA parameters optimization problem is summarized in algorithm 1. This framework applied to the path planning behavior selection problem as described above has shown positive results, which will be presented in Section 4.

We now turn our attention to the regression model implemented to learn the parameters setting in our path planning use-case. It is a multilayer Perceptron with parameters θ, comprising $L = 6$ layers (see Figure 2). Each hidden layer implements a non-linear transformation $\mathcal{H}_l(.)$, where l indexes the layer. We define the transformation $\mathcal{H}_l(.)$ as a composite function of two consecutive operations: batch normalization (BN) [7], followed by a rectified linear unit (ReLU) [5].

3 Experimental Setup

Our concept as described above is experimented at the path planning behavior selection problem, which is a fundamental challenge in robotics. We simulate an environment in Gazebo including as obstacle: a static chair, a human moving in a deterministic manner and a cat moving randomly. In this environment, a robot is given the task to reach a goal position without hitting obstacles. The rest of this section is organized as follows: it starts by formalizing the path planning problem we aim to solve, this is followed by a description of the environment in which the robot is navigating. Furthermore, our navigation framework and navigation evaluation measure is presented. Finally, attention is paid to data generation and supervised model training methodology.

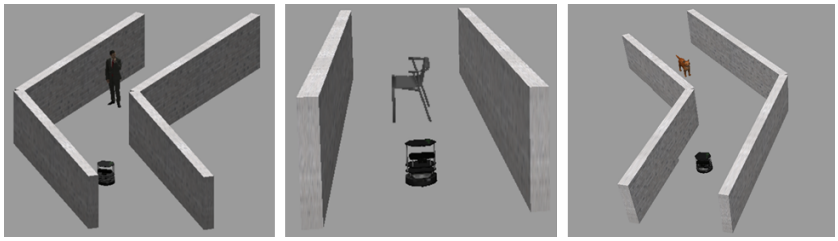

Fig. 3: Setup for experiments in simulation.

3.1 Problem formulation

Consider a mobile robot navigating in an unknown environment, we denote by $\mathcal{C} \subset \mathbb{R}^q$ the complete configuration space of the robot, where $q \in \mathbb{N}$ is the dimension of the configuration space. $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{obs}} = \mathcal{C} \backslash \mathcal{C}_{\text{free}}$ denote respectively the valid configuration in the planning space and the obstacle space consisting of robot state obstructed by collisions with either the obstacles or the environment. The robot navigates from a start state $\mathbf{q}_s \in \mathcal{C}_{\text{free}}$ with the task of reaching a goal state $\mathbf{q}_g \in \mathcal{C}_{\text{free}}$. To execute this task, the robot plans a path $\xi_{\mathbf{q}_s \to \mathbf{q}_g}$ using a path planning algorithm – a behavior selection algorithm –, which can be represented by a function $\mathcal{F}$, that solves the following motion planning problem:

$$\begin{aligned} &\xi_{\mathbf{q}_s \to \mathbf{q}_g} = \mathcal{F}(\mathbf{q}_s, \mathbf{q}_g, \mathcal{C}) \\ &\text{s.t. } \xi_{\mathbf{q}_s \to \mathbf{q}_g} = \{\mathbf{q}_1, \ldots, \mathbf{q}_m\} \\ &\text{and } \forall\ \mathbf{q}_i \in \xi_{\mathbf{q}_s \to \mathbf{q}_g}, \mathbf{q}_i \in \mathcal{C}_{\text{free}} \end{aligned} \tag{1}$$

The output of the path planning is therefore a sequence of states $\mathbf{q}_i \in \mathcal{C}_{\text{free}}$, which allow the robot to navigate without hitting obstacles. A path $\xi_{\mathbf{q}_s \to \mathbf{q}_g}$ is said to be successful if $\xi_{\mathbf{q}_s \to \mathbf{q}_g} \subset \mathcal{C}_{\text{free}}$ from the beginning to the end of the driving task. As already mentioned, path planning is an ill-posed problem, meaning that there is not a unique solution to avoid a chair for example. Our evaluation measure for comparing path solutions will be presented in Section 3.5. Furthermore, we denote by Υ_ξ the execution cost of a path $\xi_{\mathbf{q}_s \to \mathbf{q}_g}$. For a straight line between state $\mathbf{q}_a$ and $\mathbf{q}_b$ the cost of the path $\xi_{\mathbf{q}_a \to \mathbf{q}_b}$ is given by the distance: $\Upsilon_\xi(\mathbf{q}_a, \mathbf{q}_b) = \parallel \mathbf{q}_a - \mathbf{q}_b \parallel$. Since a path is the summation of consecutive small straight movements, the overall execution cost of a path $\xi_{\mathbf{q}_s \to \mathbf{q}_g} = \{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$ is then formalized as:

$$\Upsilon(\xi_{\mathbf{q}_s \to \mathbf{q}_g}) = \sum_{i=1}^{m-1} \Upsilon_\xi(\mathbf{q}_i, \mathbf{q}_{i+1}) \tag{2}$$

The path planning algorithm $\mathcal{F}$ can be configured by a set of parameter $P = \{p_1, \ldots, p_n\} \in \mathcal{P}$, where $\mathcal{P}$ is the parameter space and $p_i \in \mathbb{R}$. Let J denote the cost metrics defined for $\mathcal{F}$ when using a parameter set P in an environment

configuration $\mathcal{C}$. Our supervised learning model aims to find in a dynamic environment, for a given robot state and configuration space, a parameter set P^* that drives the robot to the goal without collision while optimizing J.

$$P^* = \arg\min_{p \in \mathcal{P}} J(\mathcal{F}(\mathcal{P}), \mathcal{C}_{\text{obs}}) \tag{3}$$

Whenever an obstacle (detected by the depth camera mounted on the robot) is encountered on a currently followed path $\xi_{\mathbf{q}_s \to \mathbf{q}_g}$, our method solves (3) finding the optimal parameter set P^* for the new configuration space that moves the robot collision-free towards the goal state $\mathbf{q}_g$.

3.2 Environment Representation

The robot is required to navigate to randomly selected target positions on an 12m x 12m, 2D map of the environment. In simulation, the environment consists of a corridor with variable width and curvature (see Figure 3). The 2D map of the environment is updated for each simulation as the corridor dimension and curvature change. The following subsections address each of the environment component individually.

Corridor. The navigation environment consists of a corridor of width w and length l. The corridor is made of 4 walls, which are pairwise parallel as shown in Figure 3. Wall 1 and 2 (the 2 top walls) have an angle α with the vertical axis, whereas walls 3 and 4 (the bottom walls) have an angle β with the same (vertical) axis. Note that the feature w, α and β are variable parameters. This allows the corridor to be modeled as: $\zeta = f(w, \alpha, \beta)$.

Obstacle. Inside the corridor, we modeled three obstacle types $O_t = \{O_1, O_2, O_3\}$ consisting of a static chair, a human character moving in a deterministic manner and a cat moving randomly (see Figure 3). During the simulation, each obstacle (except the chair, which is static) move at a constant speed O_s, where O_s is a variable parameter. Note that there is only one obstacle per simulation, that is, the cat and the human for example never appear together.

World. Joining the two later sections together, we can formally define a simulated world W as: $\mathrm{W} = f(w, \alpha, \beta, O_t, O_s)$.

3.3 Robotic platform

A kobuki based Turtlebot is used as robotic platform in simulation as well as in real world. It is equipped with a depth camera sensor mounted on it, which collects depth images in a 5 meter range, therefore allowing to sense local obstacles. The computing center of our real Turtlebot is an Intel NUC5PPYH with 8Gb RAM running Ubuntu 16.04 and the Robot Operating System (ROS).

3.4 Navigation Framework

Our navigation framework builds on the ROS navigation stack, which provides a two level motion planner consisting of a global planner and a local planner. At a time t, the global planner produces a global path (around obstacles) from the current state $\mathbf{q}_c$ to the goal state $\mathbf{q}_g$ based on the current state configuration $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{obs}}$. The local planner acts as a controller with the role of following the global path as close as possible. The ROS navigation stack is modified to use our implementation of a path planning behavior selection algorithm (RRT) as global planner. The dynamic window approach (DWA) [3] is used as default local planner in the ROS navigation stack, we do not modify this behavior. Our navigation framework therefore consists of a behavior selection algorithm (BSA) – with parameter $p_1, \ldots, p_n$ –, which plans a global path around obstacles and a local planner, which follows the planned path. If the local planner encounters an obstacle (detected by the depth camera) on the global path it is following, it aborts the process and asks the BSA for a new behavior to adopt, that is a new global path to follow. The BSA then delivers a new path (if one exists) regarding the new configuration space $\mathcal{C}$. This process is illustrated in Figure 4.

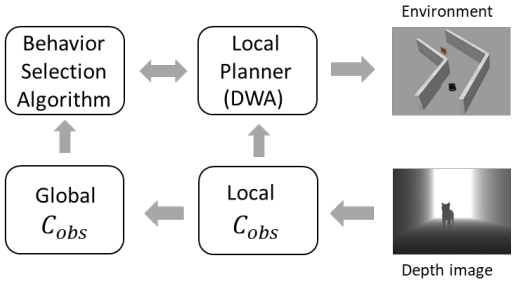

Fig. 4: Our navigation framework.

3.5 Evaluation Metric

Consider two parameter sets P_1 and P_2 of the BSA both successfully avoiding an obstacle in a navigation scenario. An important question is which of P_1 and P_2 produces the best planning strategy. To tackle this question we define the *robot collision* as metric, meaning that we expect the robot to reach the goal without colliding with obstacles. More concretely, during an entire simulation run, we compute the smallest distance $d_{\min}$ between the robot and the obstacle. A too small $d_{\min}$ implies that the avoidance maneuver was close to fail. A big $d_{\min}$ means that the robot has avoided the obstacle with a safer margin. On the other side, another requirement is that the avoidance maneuver should not move the robot too far from the shortest path leading to the goal, this is the straight line connecting start state $\mathbf{q}_s$ and goal state $\mathbf{q}_g$ assuming all obstacles are ignored.

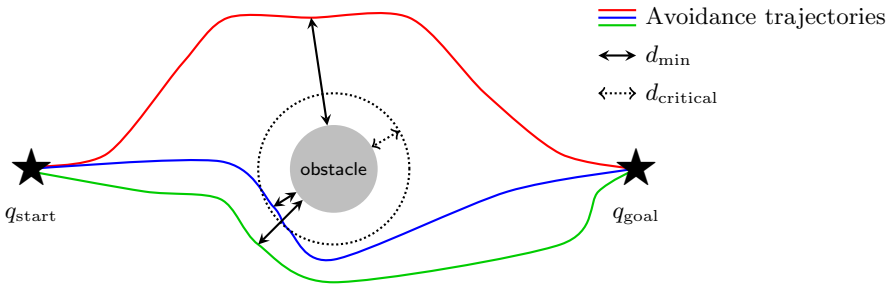

Fig. 5: Illustration of our metric. The red path avoids the obstacle with a safer margin but deviate too far from the shortest path between robot and goal states. The blue path produces the smallest $d_{\min}$ but lies inside the critical region. The green path is the one which exhibits a best trade-off between l_p and $d_{\min}$. It produces a $d_{\min}$ not deviating too much from the shortest path while yielding at the same time a safe margin to the obstacle.

This later requirement implies that a big $d_{\min}$ is also not a good choice since it deviates the robot trajectory too far from the shortest path. Denoting the length of the path produced by the avoidance maneuver as l_p, the best trajectory is therefore the one producing a path that minimizes l_p while maximizing $d_{\min}$. This trade-off between l_p and $d_{\min}$ is solved by introducing a critical distance d_{critical}, which acts as a threshold distance between the obstacle and the robot. Figure 5 illustrates the robot collision metric as described above. The red, blue and green paths show 3 different avoidance trajectories for the same environment and configuration space $\mathcal{C}_{\text{free}}$, $\mathcal{C}_{\text{obs}}$. The n best avoidance trajectories, are the one with the smallest $d_{\min} \in \mathbb{D}$ in ascending order, such that

$$\mathbb{D} = \{d^x_{\min} \geq d_{\text{critical}} \geq 0 \quad , \; x \in W\} \tag{4}$$

3.6 Data Generation

We aim at generating labelled data for the path planning BSA. The data-set consists of environment features and BSA parameters, which are the labels. To generate the data, we ran a lot of simulations, which were distributed on a computer cluster with 5 nodes. Each node consisted of a PC with Intel $i7-2600$ CPU processor clocked at 2.4 GHz, which is able to run the simulation in real time. In each simulation run, the robot is required to navigate from a start position $\mathbf{q}_s$ to a goal position $\mathbf{q}_g$ whereby the Euclidean distance between start and goal is typically 8 meter. Each simulation run takes around 40 seconds and returns a Boolean value signifying whether or not the simulation was successful. That is if the robot reached the goal state within the 40 seconds and the smallest distance between the robot and the obstacle satisfies (4). Our data generation strategy consists of three consecutive steps, which will be addressed individually in the following subsections:

Initial guess. The first step is searching for potentially good parameter settings for the path planning BSA by running a quick random search using wide ranges of parameter values.

Selective grid search. The second step performs a search using a smaller range of values centered on the best ones found during the first step. Doing so, we zoom in on a good set of planner parameters. The process is as follows: For some random generated world W, we try a grid of 90 BSA parameter configurations and choose the 3 best as the one with the smallest $d_{\min}$, whereby $d_{\min}$ satisfies (4).

Local search. The third step is a local regressor, which allows to automatically generate far more data out of the previously created database. First, we train a k-nearest neighbors multi-output regressor on the data resulting from the previous step. In a second stage, the trained KNN is used to predict the parameters setting P_i that might work well for a previously unseen world W_i. Once a parameter set producing a path satisfying (4) is found, the pair (W_i, P_i) is added to a database.

3.7 Supervised model training

We learn the parameters of the path planning BSA using the network presented in Figure 2. The training objective consists of two parts. The first part minimizes the mean-squared-error (MSE) loss between predicted parameters $\hat{P}_i$ and true parameter P_i, where i indexes the training instance. The second part accounts at each training instance for the cost $\Upsilon_i(\xi_{\mathbf{q}_s \to \mathbf{q}_g})$ of the path $\xi_{\mathbf{q}_s \to \mathbf{q}_g}$ produced by the parameters $\hat{P}_i$. This cost has been defined in (2) and is weighted in our training objective by a parameter λ, which acts as a regularization hyper-parameter. The overall loss function of the network is then computed as:

$$L(\theta) = \frac{1}{n \cdot m} \sum_{j=1}^{n} \sum_{i=1}^{m} (P_i^j - \hat{P}_i^j)^2 + \frac{\lambda}{m} \sum_{i=1}^{m} \left(1 - \frac{\Upsilon_i(\xi_{\mathbf{q}_s \to \mathbf{q}_g})}{\bar{\Upsilon}}\right)^2 \tag{5}$$

where, m is the batch size, n the number of neurons in the output layer – that is the number of parameters to predict –, and $\bar{\Upsilon}$ the length of the straight line connecting start and goal state.

4 Experimental results

We evaluate our methodology at two levels. In a first level, we want to rate the impact of using supervised learning to adjust in real-time the parameters of a robot behavior selection algorithm. For this purpose, a human solution is used as baseline. We compare the solution of a human expert to the supervised learning solution at finding parameters of the path planning BSA able to navigate the robot in the environment described in 3.2. Remember that this environment embeds three obstacle types that should be avoided: A static chair, a human moving in a deterministic manner and a random moving cat. We ran 3 type of simulations, each consisting at avoiding one of the three obstacles. Table 1 shows

that for static obstacles (here the chair), an expert human is almost as good as a supervised learning solution at providing the BSA with parameters that avoid the obstacle. For dynamic obstacles, however, the supervised learning solution dramatically improves the robots ability to safely navigate that environment. This is due to the fact that the supervised learning solution has learned to adapt the robot behavior - via the BSA parameters - to environment features such as the obstacle type, speed or trajectory.

Table 1: Baseline expert human vs supervised learning at avoiding obstacles

Obstacles type	% of success out of 1000 simulations	
	human expert	supervised learning
chair	98 %	100 %
human	56 %	96 %
cat	44 %	92 %

The second level of our methodology evaluation consists of comparing our multilayer Perceptron (MLP) learning approaches to two standard regression techniques, namely, a k-nearest neighbors with $k = 5$ and a random forest with 800 decision tree estimators. We choose those two regression techniques as they have little hyperparameters and are known to perform well on structured data and multi-regression problems. Table 2 shows that our MLP works well and is stable compared to standard regression techniques.

Table 2: Result of comparing our MLP to two supervised learning methods, when predicting parameters of BSA for the same environment configuration.

Regressor	% of success out of 1000 simulations		
	chair	human	cat
Our MLP	100 %	96 %	92 %
kNN (k=5)	99 %	95 %	79 %
random forest (800 estimators)	100 %	93 %	91 %

5 Conclusion

We have presented a method that can learn and optimize the behavior of a robot behavior selection algorithm via its parameters. The key idea was to build a supervised learner, which is trained to learn the parameters with respect to some environment features. The application of this approach to a path planning problem showed that the method is able to adjust in real-time the parameters of the algorithm therefore efficiently controlling the robot behavior while performing the navigation task. Simulations showed great results, however our method could not be extensively tested in reality due to a lack of real-world training data. In a prospective future work, we wish to empower our multilayer Perceptron with transfer learning techniques in order to forward simulation insight to reality without the need to generate a large amount of real-world data.

Acknowledgments

This work was funded by the federal state of North Rhine-Westphalia and the *European Regional Development Fund* FKZ: ERFE-040021.

References

1. Bajcsy, A., Losey, D. P., O'Malley, M. K., and Dragan, A. D. Learning from physical human corrections, one feature at a time. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction* (2018), pp. 141–149.
2. DeMers, D., and Kreutz-Delgado, K. Learning global direct inverse kinematics. In *Advances in Neural Information Processing Systems* (1992), pp. 589–595.
3. Fox, D., Burgard, W., and Thrun, S. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine 4*, 1 (1997), 23–33.
4. Frochte, J., and Marsland, S. A learning approach for ill-posed optimisation problems. In *Australasian Conference on Data Mining* (2019), Springer, pp. 16–27.
5. Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (2011), pp. 315–323.
6. Huang, Z., and Chen, Y. An improved artificial fish swarm algorithm based on hybrid behavior selection. *International Journal of Control and Automation 6*, 5 (2013), 103–116.
7. Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).
8. Izumi, K., Habib, M. K., Watanabe, K., and Sato, R. Behavior selection based navigation and obstacle avoidance approach using visual and ultrasonic sensory information for quadruped robots. *International Journal of Advanced Robotic Systems 5*, 4 (2008), 41.
9. Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*. Springer, 1986, pp. 396–404.
10. Kim, G., and Chung, W. Navigation behavior selection using generalized stochastic petri nets for a service robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 37*, 4 (2007), 494–503.
11. LaValle, S. M. Rapidly-exploring random trees: A new tool for path planning.
12. Liu, D., Cong, M., Du, Y., and Gao, S. Robot behavior selection using salient landmarks and object-based attention. In *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)* (2013), IEEE, pp. 1101–1106.
13. Murphy, T. G., Lyons, D. M., and Hendriks, A. J. Stable grasping with a multi-fingered robot hand: A behavior-based approach. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)* (1993), vol. 2, IEEE, pp. 867–874.
14. Wang, X., Ray, A., Lee, P., and Fu, J. Optimal control of robot behavior using language measure. In *Quantitative Measure for Discrete Event Supervisory Control*. Springer, 2005, pp. 157–181.
15. Wang, Y., Li, S., Chen, Q., and Hu, W. Biology inspired robot behavior selection mechanism: using genetic algorithm. In *International Conference on Life System Modeling and Simulation* (2007), Springer, pp. 777–786.