Learning-Based Auto-Tuning for Motion Controllers of Mobile Robots

JONATHAN BLIXT
Abstract

An auto-tuner of the parameters of a mobile robots motion controller is developed to improve its performance. The generality of the auto-tuner allows for similar applications on other robots or controllers. The performance is defined by a proprietary objective function for the purpose of reducing positional error, wear and energy consumption while retaining stability. The function evaluates data recorded during simulated runs of the robot. Sequential model-based Bayesian optimization (SMBO) is evaluated with different design choices. Classic black box optimizers, namely grid search, random search and Latin hypercube sampling are used as benchmarks, as well as manual tuning. It is found that SMBO performs on par with manual tuning after less than 1000 iterations searching a very large feature space, making use of no prior knowledge for setting the boundaries. When instead guided by relatively narrow boundaries around the manually chosen parameter values less than 5 iterations are needed and the performance continues improving for another 600 iterations. The SMBO demands much fewer evaluations than grid search; it outperformed grid search even when running 20 times more iterations. However, the difference in performance of different design choices for the SMBO was negligible.

Sammanfattning

Contents

1 Introduction ........................................ 1
   1.1 Control of Mobile Robots . . . . . . . . . . . . . . . . . . . . 1
   1.2 Classic Auto-Tuning Methods . . . . . . . . . . . . . . . . . . 1
   1.3 Scope of the Thesis . . . . . . . . . . . . . . . . . . . . . . . . 3
   1.4 Thesis Organization . . . . . . . . . . . . . . . . . . . . . . . . 3
   1.5 System Description . . . . . . . . . . . . . . . . . . . . . . . . 4
   1.6 Problem Formulation and Aim . . . . . . . . . . . . . . . . . . 5

2 Background .......................................... 6
   2.1 Wheel Synchronization Strategies . . . . . . . . . . . . . . . . . 6
      2.1.1 Parallel control . . . . . . . . . . . . . . . . . . . . . . . 6
      2.1.2 Master-slave control . . . . . . . . . . . . . . . . . . . . 7
      2.1.3 Cross coupling control . . . . . . . . . . . . . . . . . . . 7
      2.1.4 Circular coupling control . . . . . . . . . . . . . . . . . 8
      2.1.5 Electronic virtual line-shafting . . . . . . . . . . . . . . 8
   2.2 Learning-Based Optimization . . . . . . . . . . . . . . . . . . . 9
      2.2.1 Sequencial model-based Bayesian optimization . . . . . . 9
      2.2.2 Related work . . . . . . . . . . . . . . . . . . . . . . . . 11
   2.3 Gaussian Process . . . . . . . . . . . . . . . . . . . . . . . . . . 12
      2.3.1 Gaussian process regression . . . . . . . . . . . . . . . . 13
      2.3.2 Sparse Gaussian process regression . . . . . . . . . . . . . 15
      2.3.3 Kernels for Gaussian processes . . . . . . . . . . . . . . 15
   2.4 Acquisition Function . . . . . . . . . . . . . . . . . . . . . . . . 16
      2.4.1 Upper confidence bound . . . . . . . . . . . . . . . . . . 16
      2.4.2 Probability of improvement . . . . . . . . . . . . . . . . 17
      2.4.3 Expected improvement . . . . . . . . . . . . . . . . . . 17
      2.4.4 Designing the acquisition function . . . . . . . . . . . . . 18
      2.4.5 Maximizing the acquisition function . . . . . . . . . . . 18

3 Method .............................................. 19
   3.1 The Choice of Optimizers to Investigate . . . . . . . . . . . . . . 19
   3.2 Implementation using GPyOpt Package . . . . . . . . . . . . . . 20
   3.3 Test Performance in Simulator . . . . . . . . . . . . . . . . . . . 21
      3.3.1 Different boundaries . . . . . . . . . . . . . . . . . . . . 21
      3.3.2 Different probabilistic models . . . . . . . . . . . . . . . 22
      3.3.3 Different acquisition functions . . . . . . . . . . . . . . 22
      3.3.4 Defining the objective function . . . . . . . . . . . . . . 22
   3.4 Comparing SMBO to other Tuning Approaches . . . . . . . . . . 23
      3.4.1 Standard grid search and Latin Hypercube sampling . . . 23
      3.4.2 Random search . . . . . . . . . . . . . . . . . . . . . . . 24
      3.4.3 Manual tuning . . . . . . . . . . . . . . . . . . . . . . . 24

4 Results ............................................. 25
   4.1 Volatility of Scores for Same Input . . . . . . . . . . . . . . . . 25
   4.2 Different Probabilistic Models . . . . . . . . . . . . . . . . . . 26
   4.3 Different Acquisition Functions . . . . . . . . . . . . . . . . . 27
   4.4 Different Boundaries . . . . . . . . . . . . . . . . . . . . . . . 28
   4.5 Performance of Different Optimizers . . . . . . . . . . . . . . . 31
5 Discussion 33
6 Conclusion 36
A Warped Gaussian process regression 40
B Random Forest 41
1 Introduction

The field of robotics has quickly gained a tremendous importance as their versatility and areas of application are increasing. Sales of industrial robots have increased by 14% globally on average each year since 2009 [1]. Thanks to technological advances, the industry is heading from stationary manipulators towards mobile robots. An example of this is the company Ocado that specializes in online retail with home deliveries straight from their warehouses. Those warehouses are highly automated with hundreds of mobile robots navigating a grid full of groceries. The robots operate on tracks at high speeds with small margins, hence an accurate control system is of essence to avoid derailing, crashes and minimize wear and energy consumption.

1.1 Control of Mobile Robots

To start with, the word robot does not have a clear universal definition. However, few would probably disagree with the statement that in general a robot is a machine that performs an intelligent connection between perception and action. This connection requires the use of sensors and actuators. In contrast to static manipulators, mobile robots have the additional capability of locomotion, i.e. moving from one location to another, not just moving a manipulator. There are a number of ways of getting around, such as walking and flying, but the simplest, cheapest, most efficient and common is by wheeled locomotion.

There are plenty of different control approaches to wheeled locomotion with different degrees of freedom. Non-holonomic mobile robots operating on planar surfaces often have two degrees of freedom; speed and heading. Examples are robotic vacuum cleaners and autonomous cars, ignoring the suspension. The autonomous warehouse robots considered in this report only have one degree of freedom since they are guided by tracks. This means that steering, i.e. changing the heading, is not part of the controllers objective. However, the controller must handle multiple wheels simultaneously, namely four for each of the two orthogonal directions of the grid. This demands a speed synchronization technique among all wheels that can handle unbalanced loads, loss of adhesion and disturbances. Five such synchronization controllers will be introduced in the background, section 2.1. Those are parallel-, master-slave-, cross coupling-, circular coupling- and electronic virtual line-shafting control, which are typical control strategies commonly used in industry and research [2–5].

1.2 Classic Auto-Tuning Methods

To begin with, some classic methods of tuning are mentioned. Perhaps the most intuitive yet widely used tuning technique for controllers is the manual method of trial and error where analysis and expertise can give a good starting point and possibly some intuition of what parameters to try next. This is simply referred to as manual tuning. To do any kind of tuning, or optimization, the parameter space demands set boundaries. This necessitates some human input, ranging from enormous parameter spaces with limits set by basic knowledge to very tight boundaries derived by theory, analysis and experience. Once the boundaries are set, however, the process of choosing what parameter sets to
evaluate can be automated. Two classic auto-tuning methods, grid search and random search allow this automation. In order for them to find the optimal parameter set one has to define the desired properties of the result. This is done through the objective function, or scoring function. Some properties can be difficult to define in mathematical terms, like well behaving, a reason to chose manual tuning if you e.g. prefer looking at the resulting graphs to give your subjective opinion. On the other hand, defining this function gives an accurate, objective and repeatable outcome that alleviates the need for analysis of every iteration.

Grid search is the method of dividing the parameter space into a grid with set limits and searching by brute force for the optimal value, without making use of the information gained so far. The parameters have to be discrete or continuous but discretized to be able to define a grid. The grid can have any dimension which is given by the number of parameters and it is often, but not necessarily, equidistant. A grid search is exhaustive, meaning it will test every combination in the grid. However, when dealing with continuous parameters the number of discretized points, or resolution, can always be increased to improve the chance of finding a minimum. This implies that it is impossible to guarantee finding an optimal continuous parameter set using grid search since the optimal parameters could always have values in between grid nodes. The grid may also be defined using Latin hypercube sampling as explained in section 3.4.1, but in that case the grid will not be sampled exhaustively.

Random search follows no grid, instead each parameter’s value is randomly chosen within its boundaries and combined into one parameter set. However, the random sampling is not restricted to being uniform which allows for inclusion of prior knowledge in the form of designing sampling probability distributions.

Figure 1 shows a comparison between standard grid search, random search and latin hypercube sampling. In the example shown 9 samples of a two dimensional feature space are gathered using each method. Grid search and random search belong to the class of uninformed optimizers since non of the knowledge gained by samples from previous iteration is used to make an intelligent decision on where to sample next. Methods that do consider gained knowledge are referred to as learning-based optimizers and will be introduced in section 2.2.
Figure 1: Examples of gathering 9 samples using the three different uninformed sampling techniques standard grid search, random search and Latin hypercube sampling. The curves on top and to the left symbolize a cross section of the objective score to indicate the influence each of the two parameters have on the result.

1.3 Scope of the Thesis

The auto-tuner developed will be based on expensive black box optimizers. Of those optimizers standard grid search, Latin hypercube sampling and random search are the simple methods used mainly as benchmark for the learning based optimizers investigated. The learning based optimizers are based on sequential model-based Bayesian optimization (SMBO) with different choices of probabilistic models, acquisition functions and boundaries. The probabilistic models studied are Gaussian process (GP), sparse GP, GP with Markov chain Monte Carlo acquisition, (output-) warped GP, input warped GP and random forest. Unfortunately the last three could not be evaluated due to implementation errors in the optimization package GPyOpt and their presentations were consequently put in the appendix. The acquisition functions studied and evaluated are expected improvement, probability of improvement and upper confidence bound.

1.4 Thesis Organization

The classic auto-tuning approaches introduced in section 1.2 of the introduction will be used as benchmarks for the learning-based optimizer presented in section 2.2.1 of the background. This optimizer, SMBO, will be investigated thoroughly by presenting the different design choices listed in the scope of the thesis 1.3. The method of developing the auto-tuner for this application is presented in section 3 where the choice of optimizers to investigate is argued for in section 3.1. After that, the implementation is described in section 3.2 followed by a description of the simulator used, and how it was used, in section 3.3. Other related methods of auto-tuning are presented in section 2.2.2 as well as a possible extension of the
SMBO in section 2.2.2. The results are then presented in section 4, structured the same way as the method section, and is followed by a discussion in section 5 and finally the conclusion in section 6.

1.5 System Description

The learning-based auto-tuner is developed for the purpose of improving the performance of autonomous warehouse robots. The tuner should, however, be as generic as possible to be applicable to different control systems and robot versions.

The overall system consists of an aluminum grid with each cell storing multiple plastic crates stacked on each other. See Figure 2 for a picture from within one of these warehouses. The autonomous robots move on top of this grid, picking up and dropping of the crates filled with groceries. If the desired item is in a crate below other crates the robots will cooperate to have the crates on top removed. The swarm of identical robots is controlled by a 4G network system running algorithms to optimize the workflow and avoid collisions.

Figure 2: One of Ocado’s autonomous grocery picking systems. Each cell is filled with stacks of plastic crates that can be picked up by the robots running on tracks above. [6]

The robots are equipped with eight wheels, two on each side, to be capable of maneuvering the two dimensional grid. Two pairs of wheels on opposite sides are in contact with the grid track at any given time while the other two pairs are hoisted. The motors and control systems for both four wheel configurations are identical. Hence the motion control can be simplified to four wheels driving in one direction. It should, however, be mentioned that the robot and grid cells are not square which means that the distance between the front wheels and the back wheels are different for the different directions. The robots operate at a top speed of 4 m/s with a maximal acceleration of 2 m/s² including regenerative
braking.

An issue that should be avoided by a good choice of parameters is oscillation. Because the grids can be several hundred meters long in both directions oscillations could grow to such a magnitude as to derail the robots.

1.6 Problem Formulation and Aim

The problem spurring this research is the difficult and demanding task of setting the parameters of a complex system of controllers. The motion controller of the autonomous warehouse robots receive position, velocity and acceleration profiles as reference which are tracked by an elaborate structure of subcontrollers. Many analysis tools can be used to find parameters for individual controllers, but already at this stage tuning of the parameters is required to account for error in assumptions, mechanics and electronics if high accuracy is demanded. Coupling controllers into a complex system further increases the importance of tuning over analysis.

The aim of this study is to design an auto-tuner that outperforms manual tuning. The performance of the auto-tuner will be measured by the number of iterations needed to reach a satisfactory performance of the motion controller for the robots. An objective function defining the performance of the motion controller needs to be developed to do so.

In this work, we do not restrict ourselves to a specific set of parameters, instead the approach developed should be applicable to any given set of parameters. The main approach provided will be based on machine learning in an effort to minimize the parameter evaluations needed.
2 Background

First, several different wheel synchronization strategies will be presented in section 2.1 to give some background on the application area studied for the auto-tuner developed in this work. After that the main optimizer studied and evaluated for the auto-tuner is presented in section 2.2.1. This is followed by a deep dive into the two main probabilistic models used for SMBO namely Gaussian processes in section 2.3 and Random forest in section B. Finally the acquisition functions, used in this work to choose what parameter set to evaluate each iteration by the SMBO, are presented in section 2.4.

2.1 Wheel Synchronization Strategies

A few common approaches to wheel speed synchronization are presented in the following subsections. It is important to notice that the same approaches can be used to synchronize other physical quantities such as the rotational angle or, in other words, integrated angular speed. This is useful for mobile robots that ought to move straight but lack mechanical connections between the different wheels. These control strategies are the bare minimum configurations but can be extended and puzzled together into complex systems. Again, it should be stressed that the auto-tuner should be indifferent to the controller used, what happens during the simulation is treated as a black box.

2.1.1 Parallel control

One of the simplest strategies for synchronizing the speed of two or more motors is by giving the individual closed loop systems the same reference signal. The different motor controllers work in parallel towards the same goal, hence the name Parallel control. It is also referred to as Master reference control since all motors follow a master reference without considering the other motors. A block diagram of the basic, generic parallel controller is found in Figure 3. This type of controller is completely decoupled and any unbalanced load, noise or electromechanical difference between the motors would destroy the synchronization. One advantage of this simple approach is the motors’ quick rejection of measured disturbances in the output angular speed; its strength is in tracking performance at the cost of poor synchronization [2].

![Figure 3: Parallel Control](2.png)
2.1.2 Master-slave control

The master-slave configuration can be seen in Figure 4. Here only one motor controller tracks the input reference, the master, and the other motor controllers, the slaves, follow the master by its output angular speed. This synchronization is direct in the sense that the slaves are tracking, or designed to sync with, the master. This improves the synchronization between the motors since the slaves will react to disturbances affecting the master. However, the synchronization is unidirectional meaning neither the master, nor the other slaves, will react to disturbances in a slave controller. This issue could be problematic in applications such as rollers conveying fragile objects; if a slave experiences an increased load decreasing its speed the other rollers will continue tracking a higher speed which could tear the object apart. On the other hand, if one motor has a much larger moment of inertia than the other choosing it as the master should let the motors synchronize well following a stable reference signal [2].

Figure 4: Master-slave control [2, p. 2115]

2.1.3 Cross coupling control

Cross coupling control can be viewed as an extension of the parallel control configuration since the closed loops have a master reference as input but where the feedback includes the relative error between the motor speeds. A schematic of the control strategy can be found in Figure 5. The configuration can either be symmetric or asymmetric with different values on the relative error gain and different controllers. The coupling allows both motors to react to disturbances in either output motor speed, in contrast to the master-slave strategy. This means that an error in one output speed is corrected by both motors. This considerably reduces the error in heading for differential drive mobile robots which is the most significant error in path following. The effectiveness in reducing internal errors such as differences in friction and parameters of the drive loops has been verified experimentally [7]. The strength of the coupling, or rather the compromise between tracking the master reference and the neighbouring motor, can be adjust by the coefficients $K_1$ and $K_2$ in Figure 5. To reach synchronization between the motors, these gains need to be big which has the drawback of amplifying noise in the output causing torque ripples [4]. Cross coupling control cannot easily be extended to systems of more than two motors due to difficulties in deriving appropriate feedback gains for multiple relative errors [2].
2.1.4 Circular coupling control

Circular coupling control, also known as ring coupling control, is a strategy that can easily be extended to a system of an arbitrary number of motors, in contrast to cross-coupling control. This is expressed by the dotted lines in Figure 6, where any number of additional motor controllers can be inserted following the same pattern. Each motor controller, call it $i$, is fed with the reference error $e_{ref,i}$ and the relative error $e_{i,i+1}$ between itself and its nearest neighbour in a circular pattern. The last controller can be fed with the output of the first to close the circle. It has been shown that not only is the synchronization improved considerably, compared to the master-slave configuration, but the reference tracking error is also decreased and the robustness improved [8].

2.1.5 Electronic virtual line-shafting

One intuitive approach to synchronizing drive loops is to mimic mechanical synchronization through drive shafts, chains etc. One such strategy is electronic
virtual line-shafting (EVLS). EVLS simulates a mechanical shaft by calculating the torque produced by a real shaft given the difference in angle and angular speed. Introduce the damping gain $b_r$, the stiffness gain $K_r$ and the integrated stiffness gain $K_{ir}$ and the resulting restoring torque $T_r$ is calculated as

$$T_r = b_r(\omega_2 - \omega_1) + K_r(\theta_2 - \theta_1) + K_{ir}\int(\theta_2 - \theta_1)dt.$$  \hspace{1cm} (1)

The first two terms have their physical analogues while the integrated stiffness gain is introduced to maintain zero relative displacement at non-zero load torque [10]. A schematic of the controller is shown in Figure 7 without the integrated stiffness gain. As can be seen in the figure both drive loops calculates their corresponding restoring torque and the signals are added and fed back to the outer loop.

![Figure 7: Electronic virtual line-shafting [2, p. 2115]](image)

### 2.2 Learning-Based Optimization

The classical, uninformed auto-tuners introduced in section 1.2 have been joined by more modern approaches involving learning. Among all learning-based optimizers sequential model-based Bayesian optimization was chosen as the best candidate for this application by the reasons provided in section 3.1.

#### 2.2.1 Sequential model-based Bayesian optimization

Bayesian global optimization or the random function approach introduces stochastic processes to the field of global optimization. It dates back to 1964 when it was presented by Harold Kushner [11]. He described it as a versatile and practical method since it can handle noisy data [12].

Bayesian learning-based global optimizers are well suited for black box functions that are expensive to evaluate. An example of expensive black box functions is simulation based evaluations, which is dealt with in this work. A probabilistic model is created to be evaluated considerably quicker and work as a surrogate for the true function to optimize; the black box function. It is also easy to make
use of expertise or previous experience by designing the prior beliefs to reflects this information.

Building a probabilistic model iteratively based on Bayesian learning and using it to pick the next point to evaluate is called Sequential Model-based Bayesian Optimization (SMBO). SMBO can be viewed as belonging to the field of supervised machine learning. The supervision comes from the evaluations of the black-box function, each score treated as a label of the corresponding parameter set. The SMBO is taught how to build its surrogate model by input-output pairs from the simulation or, indeed, each run of the real robot, whichever is treated as the black-box function to evaluate.

The generic algorithm is presented as pseudo code in algorithm 1. It is initiated by evaluating \( t \) different parameter sets \( \theta_{1:t} \). This can be done completely random or using Latin hypercube sampling for example, see section 3.4.1. Those are evaluated using the utility function, \( U(\theta_i) \). The utility, or objective-, function in this case demands measurements from a simulation or the real robot because it is the performance of the robot that ought to be improved. The utility scores \( y_i \leftarrow U(\theta_i) \) paired with their parameter sets \( \theta_i \), are then used to fit a probabilistic model \( p_M(y|\theta) \) showing the continuous probability distribution of scores to expect for every parameter set. Now, since uncertainty is involved, one has to weigh the desire to get a specific score with the probability of that score. This is done by taking the argmax of the acquisition function \( a(\theta, p_M(y|\theta)) \). The acquisition function is continuous and one-dimensional which means that an abundance of simple regressors can be used to find the argmax in negligible time. The acquisition function is an informed probabilistic representation of the objective function built, see section 2.4.

Now rerun the simulation with the parameter set \( \theta_j \) that maximizes this function and iterate \( T \) times. Other stopping criteria could be used. Finally, return the parameter set \( \theta^* \) that gave the best score, not necessarily the last parameter set evaluated. The choice of probabilistic model \( p_M(y|\theta) \) could be different kinds of Gaussian processes, section 2.3, or Random Forest, section B.

**Input:** Target/Objective/Utility function \( U \); Number of iterations \( T \); Parameter space \( \theta \); Initial parameter sets \( \theta_{1:t} \)

**Result:** The best parameter set \( \theta^* \) found

for \( i \leftarrow 1 \) to \( t \) do
  \( y_i \leftarrow \text{Evaluate } U(\theta_i) \)
end

for \( j \leftarrow t + 1 \) to \( T \) do
  \( p_M(y|\theta) \leftarrow \text{Fit probabilistic model on performance data } (\theta_i, y_i)_{i=1}^{t+1} \)
  Select \( \theta_j = \text{argmax}_{\theta \in \Theta} a(\theta, p_M(y|\theta)) \)
  \( y_j \leftarrow \text{Evaluate } U(\theta_j) \)
end

return \( \theta^* \in \text{argmin}_{\theta_j \in \{\theta_1, \ldots, \theta_T\}} y_j \)

**Algorithm 1:** Generic Sequential Model-based Bayesian Optimization. SMBO(\( U, T, \theta, \theta_{1:t} \)) [13]
2.2.2 Related work

Bayesian Optimization for Learning Gaits under Uncertainty [14]

The study was performed by Roberto Calandra et al. on a bipedal robot with actuated hip and knee joints and spring equipped lower legs. This application is very similar to our in the sense that the main objective is to limit the number of experiments, or real world runs, needed to find optimal parameters. They study only a Gaussian Process with covariance kernel given by a combination of a Squared Exponential and Gaussian noise.

\[
k(x_p, x_q) = \sigma_f^2 \exp \left( -\frac{1}{2}(x_p - x_q)^T D^{-1}(x_p - x_q) \right) + \sigma_\epsilon^2 \delta_{x_p, x_q},
\]

\(D = \text{diag}(l_1^2, ..., l_D^2)\) (2)

But they look at both manually and automatically choosing the hyperparameters \(\sigma_f, \sigma_\epsilon\) and \(l_1^2, ..., l_D^2\). The robot is controlled by a finite state machine where the transitions between the four states are controlled by threshold values on the joint angles. The control parameters to be tuned are these four thresholds as well as the four control signals to each joint. The acquisition functions studied were PI, EI and UBC. A comparison between SMBO, grid search and random search was also made, but they only investigated manual tuning of the hyperparameters of the SMBOs, not the control parameters themselves. See Figure 8 for the main results.

![Figure 8: Results from study by Roberto Calandra et al. [14]. All SMBOs, here abbreviated BO, are modeled using GP regression. GP-UCB has its hyperparameter \(\kappa\) automatically chosen according to equation (21).](image)

Practical Bayesian Optimization of Machine Learning Algorithms [15]

In this study Jasper Snoek et al. investigate the effect the design choices of the GP has on the performance of the SMBO. The design choices of interest are the covariance function and their hyperparameters, the acquisition function and the objective function. A fully Bayesian treatment of the hyperparameters is proposed. Instead of finding a single set of hyperparameters by some criterion, every choice of hyperparameters is considered by integrating the acquisition function over the hyperparameters. This is described further in section 2.4.4.

Using Deep Belief Nets to Learn Covariance Kernels for Gaussian Processes [16] Many choices can be made when implementing SMBO. Those
include, but are not limited to, the choice of surrogate model and acquisition function. Even when the decision has been made on what type of surrogate model will be used, there might still be more design choices to make within that class. For Gaussian processes the choice must be made on how to model the covariance between data point. This comes down to designing a Kernel as discussed in section 2.3.3.

All these design choices for the auto-tuner can be seen as parameters that may also be optimized for the application at hand. This makes it possible to auto-tune the auto-tuner, but then when do we stop? The process of auto-tuning the covariance kernel for Gaussian processes has already been studied, to name one example. Ruslan Salakhutdinov and Geoffrey Hinton [16] have shown promising results tuning

\[ K(x_p, x_q) = \alpha \exp \left( -\frac{1}{2\beta^2} | F(x_p | W) - F(x_q | W) |^2 \right) \]  

(3)

where \( F(x | W) \) is a multi-layer, non-linear mapping to a feature space, parameterized by \( W \). This mapping is learned by using Deep Belief Networks.

### 2.3 Gaussian Process

Gaussian probability distributions are well known but the theory may also be extended to not only apply to random variables but to functions as well. Discretized finite functions could be viewed as vectors containing a function value for each input. If this connection between functions and sets of variables is made the following definition can be applied to functions as well. **Gaussian Process**

A Gaussian process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution. [17] Here the collection of random variables can be viewed as a discrete probabilistic function. To put it in other words, instead of a function \( f \), evaluated at \( x' \), giving a deterministic value \( f(x') = y' \) it returns a random variable according to a Gaussian distribution

\[ f(x') \sim \mathcal{N}(\mu, \sigma) \]  

(4)

with \( \mu \) and \( \sigma \) representing the mean and standard deviation respectively. Not only is the evaluation of a single point a random variable, but as is expressed in the definition 2.3, any subset of the collection is jointly distributed. The joint distribution between any subset of the GP is expressed in equation (5). For clarity the joint distribution for single samples \( f(x) \) and \( f(x') \) is shown first with the mean chosen to be zero. The scalar \( k(x, x') \) is the covariance between \( x \) and \( x' \). Then the joint distribution between sets of multiple samples is shown where several vectors \( x \) are concatenated into a matrix \( X \). Similarly the vector \( f \) and matrix \( K \) are concatenations of the corresponding \( f \)'s and \( k \)'s. Each \( K \) is the covariance matrix of every pair of variables from the sets indicated by the arguments. The mean is assumed to be zero without loss of generality.
\[
\begin{bmatrix}
    f(x) \\
    f(x')
\end{bmatrix} \sim \mathcal{N}
\begin{pmatrix}
    0, \\
    [k(x, x') \\ k(x', x')]
\end{pmatrix}
\]
\[
\begin{bmatrix}
    f(X) \\
    f(X')
\end{bmatrix} \sim \mathcal{N}
\begin{pmatrix}
    0, \\
    [K(X, X) \quad K(X, X') \\ K(X', X) \quad K(X', X')]
\end{pmatrix}
\]

\[x \in X, \quad x' \in X', \quad k \in K, \quad f \in f\]

2.3.1 Gaussian process regression

Gaussian process regression, or kriging, is a method of building a model based on discrete data points by interpolation.

Basing the machine-learning on Gaussian processes not only allows for computation of the posterior in closed form, but also gives an estimate of the uncertainty for each predicted output. Regression is the subfield of supervised learning that deals with prediction of continuous quantities, in contrast to classification that deals with discrete labels.

Under the Gaussian process model the prior is Gaussian. The initialization can have different designs such as sampling predetermined points, randomly or using prior knowledge to some extent. The Gaussian process model generally depends on a few so called hyperparameters. The term hyperparameters is used for parameters that are constant and set before, and not as part of, the learning phase. However, these can often be set automatically. This is usually done by maximizing the likelihood of the output given the input. The need of setting hyperparameter values is due to the fact that the covariance functions used will typically have some free parameters. Recall the covariance function \( k \) of expression (5). For example, the squared-exponential covariance function in one dimension with Gaussian noise has the following form.

\[
k_{SE}(x_p, x_q) = \sigma_f^2 \exp \left(-\frac{1}{2\ell^2}(x_p - x_q)^2\right) + \sigma_n^2 \delta_{pq}
\]

That is, for this covariance function the hyperparameters are the length-scale \( \ell \), the signal variance \( \sigma_f^2 \) and the noise variance \( \sigma_n^2 \). As already indicated, these can be set by the criterion of optimizing the marginal likelihood, or similar. See section 2.4.4 for more on designing the covariance function as part of the acquisition function.

Rasmussen and Williams at MIT [17] explain Gaussian Process Regression, or Kriging, with two different approaches. The Weight-space view extends the concept of linear regression with weights \( \mathbf{w} \) and Gaussian noise \( \epsilon \).

\[
f(x) = \mathbf{x}^T \mathbf{w}, \quad y(x) = f(x) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_n^2)
\]
Given this model and \( N \) input output pairs \( \{ \mathbf{x}_i, y_i \}_{i=1}^N \), where the column vector inputs are concatenated into a matrix \( \mathbf{x} \), the likelihood becomes

\[
p(y|\mathbf{X}, \mathbf{w}) = p(\{y_i\}_{i=1}^N|\{\mathbf{x}_i\}_{i=1}^N, \mathbf{w}) = \prod_{i=1}^N p(y_i|\mathbf{x}_i, \mathbf{w}) =
\]

\[
= \prod_{i=1}^N \frac{1}{\sqrt{2\pi\sigma_n}} \exp\left(-\frac{(y_i - \mathbf{x}_i^T \mathbf{w})^2}{2\sigma_n^2}\right) =
\]

\[
= \frac{1}{(2\pi\sigma_n^2)^{N/2}} \exp\left(-\frac{1}{2\sigma_n^2}|y - \mathbf{X}^T \mathbf{w}|^2\right) = \mathcal{N}(\mathbf{X}^T \mathbf{w}, \sigma_n^2 \mathbf{I}).
\]

If a prior over the weights \( p(\mathbf{w}) \) is specified the marginal likelihood can be computed as

\[
p(y|\mathbf{X}) = \int p(y|\mathbf{X}, \mathbf{w}) p(\mathbf{w}) d\mathbf{w}
\]

(9)

giving rise to a posterior distribution over the weight through Bayes’ rule

\[
p(\mathbf{w}|y, \mathbf{X}) = \frac{p(y|\mathbf{X}, \mathbf{w}) p(\mathbf{w})}{p(y|\mathbf{X})}.
\]

(10)

In summary, given a prior belief of the weight of the model, and observed noisy input-output pairs, the weight of the model can be updated using equation (10). Since this method of Bayesian Learning does not output a single set of weights \( \mathbf{w} \), but rather a distribution over weights, the best prediction of the black box function \( f \) evaluated at \( \mathbf{x}' \) accounts for the entire distribution by integrating over the weights and their probability.

\[
p(f(\mathbf{x}')|\mathbf{x}', \mathbf{y}, \mathbf{X}) = \int p(f(\mathbf{x}')|\mathbf{x}', \mathbf{w}) p(\mathbf{w}|\mathbf{X}, \mathbf{y}) d\mathbf{w} = \mathbf{x}'^T \int w p(\mathbf{w}|\mathbf{X}, \mathbf{y}) d\mathbf{w}
\]

(11)

If the Gaussian prior has variance \( \Sigma_p \) and zero mean the distribution over function values at \( \mathbf{x}' \) is explicitly

\[
f(\mathbf{x}') \sim \mathcal{N}(\frac{1}{\sigma_n^2} \mathbf{x}'^T \mathbf{A}^{-1} \mathbf{X} \mathbf{y}, \mathbf{x}'^T \mathbf{A}^{-1} \mathbf{x}')
\]

\[
\mathbf{A} = \frac{1}{\sigma_n^2} \mathbf{X} \mathbf{X}^T + \Sigma_p^{-1}.
\]

(12)

The method can now be extended to incorporate non-linearities by mapping the inputs into a feature space of choice, call it \( \phi(\mathbf{x}) \). One example is a feature space of powers, such as \( \phi(\mathbf{x}) = (1, \mathbf{x}^2, \mathbf{x}^3) \), allowing for polynomial regression. Our model in equation (7) now becomes

\[
f(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w}, \quad y(\mathbf{x}) = f(\mathbf{x}) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_n^2).
\]

(13)

By introducing \( \Phi = \Phi(\mathbf{X}) \) as the concatenated mapping \( \phi(\mathbf{x}_i) \) of all \( \mathbf{x}_i \) in \( \mathbf{X} \) and the covariance matrix \( \mathbf{K} = \Phi^T \Sigma_p \Phi \) formula (12) becomes

\[
f(\mathbf{x}') \sim \mathcal{N}(\Phi^T \Sigma_p \Phi(\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y}, \Phi^T \Sigma_p \Phi^T - \Phi^T \Sigma_p \Phi(\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \Phi^T \Sigma_p \Phi^T).
\]

(14)
The second, alternative approach to GP regression, mentioned by Rasmussen and Williams, is the Function-space View. Let us consider the inferences directly in the function space.

\[
m(x) = \mathbb{E}[f(x)] \\
\begin{align*}
k(x, x') &= \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))]
\end{align*}
\]  

Substituting the function for our model in equation (13) and using the same zero mean prior as before the mean and covariance becomes

\[
m(x) = \Phi(x)^T \mathbb{E}[w] = 0 \\
k(x, x') = \mathbb{E}[f(x)f(x')] = \Phi(x)^T \mathbb{E}[ww^T] \Phi(x') = \Phi(x)^T \Sigma_p \Phi(x')
\]  

Recalling definition 2.3 thus shows that this model defines a GP with \( f(x) \) and \( f(x') \) jointly Gaussian with mean and covariance given in equation (16). Instead of designing \( \Phi \) and \( \Sigma_p \) separately the covariance kernel model \( k(x, x') \) will be chosen as one, see section 2.3.3.

Now to find the marginal likelihood

\[
p(y|x) = \int p(y|f, x)p(f|x)df
\]

that describes the distribution of scores expected for given input parameter sets. As stated in the beginning of this section the prior of a GP is Gaussian

\[
p(f|X) \sim \mathcal{N}(0|K),
\]

which, taking the logarithm and using equation (17), yields

\[
-\log p(y|x) = \frac{1}{2}(y^T(K + \sigma_n^2 I)^{-1}y + \log |K + \sigma_n^2 I| + n \log 2\pi).
\]

2.3.2 Sparse Gaussian process regression

An issue with the standard, exact, implementation of GP regression is its computational cost for high numbers \( N \) of training samples. In fact, the calculation time scales as \( O(N^2) \) and the necessary memory with \( O(N^3) \) [18, 19]. This has sprung the development of more computationally efficient GP regression, namely the approximate Sparse GP regression. It is based on reducing the computational cost of inverting the full rank matrix \( K + \sigma_n^2 I \) of equation (19), call it \( K_{ff} \), by the low rank approximation \( Q_{ff} = K_{fu}K_{uf}^{-1}K_{uf} \). There are several methods of producing this decomposition where a \( K_{uu} \) of arbitrary size \( M \) can be calculated resulting in a reduction of the computation time to \( O(NM^2) \) and memory needed to \( O(NM) \). The most common approaches are DTC [20] and FITC [21].

2.3.3 Kernels for Gaussian processes

In order to compute the covariance between data points in a Gaussian process a kernel needs to be chosen. A list of kernels commonly used as covariance functions is provided in Table 1.
Kernel | Definition
--- | ---
Constant kernel | \( k(x_p, x_q) = C, \ C \in \mathbb{R} \)
Gaussian Noise kernel | \( k(x_p, x_q) = \sigma^2 \delta_{x_p, x_q} \)
Finite dimensional linear kernel | \( k(x_p, x_q) = x_p^T x_q \)
Spherical Gaussian kernel (RBF) | \( k(x_p, x_q) = \exp\left(-\frac{1}{2}(x_p - x_q)^T D^{-1}(x_p - x_q)\right), \quad D = \text{diag}(l_1^2, \ldots, l_D^2) \)
Squared Exponential kernel | \( k(x_p, x_q) = \exp\left(-\frac{|x_p - x_q|^2}{2 \alpha l^2}\right) \)
Matern kernel | \( k(x_p, x_q) = 2^{\frac{\nu - 1}{2}} \Gamma\left(\nu\right) B_\nu r^{\nu} B_\nu\left(r/l\right) \)
Rational Quadratic kernel | \( k(x_p, x_q) = (1 + \frac{|x_p - x_q|^2}{2 \alpha l^2})^{-\alpha}, \quad \alpha \geq 0 \)
Periodic kernel | \( k(x_p, x_q) = \exp(-\frac{2 \sin^2(\pi |x_p - x_q|/l)}{l}) \)

Table 1: Examples of covariance kernels. \( \Gamma \) is the gamma function, \( B_\nu \) the modified Bessel function of the second kind and \( l \) the length scale considered.

In the expression for the Matern kernel the \( \Gamma \) is simply the gamma function and \( B_\nu \) is the modified Bessel function of the second kind. With increasing \( \nu \) the sample path of the Matern kernel becomes smoother and converges to the Squared Exponential kernel, also called Radial-Basis Function (RBF) kernel. The same is true for the Rational Quadratic kernel as \( \alpha \) goes to infinity.

It is important to point out that when calculating the distance between the data points you can always use different length scales, hence the \( l \) in almost every kernel. The length scales may also be different for different dimensions.

2.4 Acquisition Function

You could greedily choose the parameter that maximizes the score of your surrogate model, but then you do not consider uncertainty. There might be a part of the model that is very uncertain, which means that there is a possibility that the scores generated by that parameter set are worse than expected. We do not simply want to exploit, but also explore.

2.4.1 Upper confidence bound

Probably the simplest way of considering both exploitation and exploration in the acquisition function is taking the mean \( \mu \) and adding the uncertainty, more precisely the standard deviation \( \sigma \), with some weighing factor \( \kappa \)

\[
UCB(\theta) = \mu(\theta) + \kappa \sigma(\theta)
\]

(20)

The factor \( \kappa \) can automatically be chosen as [22]

\[
\kappa = \sqrt{2 \log \left( \frac{n^{d/2+2\pi^2}}{3\delta} \right)}, \quad \delta \in [0, 1]
\]

(21)

where \( n \) is the current iteration and \( d \) is the dimension of the parameter space. This choice of \( \kappa \) gives a small bounded regret, derived by Srinivas et. al. [22],

16
with probability $1 - \delta$. If the objective is to minimize the acquisition function $\kappa$ is chosen to be negative and the function is referred to as the lower confidence bound.

### 2.4.2 Probability of improvement

When trying to find the optimal parameter set it is intuitive to compare the score of the parameter set to the score of the best parameter set $\theta^*$ found so far. The (Maximum) Probability of Improvement (PI) is defined as

$$
\Phi\left(\frac{\mu(\theta) - f(\theta^*)}{\sigma(\theta)}\right),
$$

where $\Phi$ is the cumulative normal distribution function which is monotonically increasing and hence can be maximized by simply maximizing

$$
\frac{\mu(\theta) - f(\theta^*)}{\sigma(\theta)}.
$$

Naturally, for the probability of improvement to be high we look for a high mean value with high certainty, hence the variance in the denominator. This is, however, very different from UCB where uncertainty is sought after.

### 2.4.3 Expected improvement

The Expected positive Improvement (EI) considers all possible scores of a parameter set given the possible outcomes according to the probabilistic model. It is simply the expected increase in score compared to the best found yet, ignoring all lower scores.

$$
EI(\theta) = \mathbb{E}\{\max(f(\theta^*) - f(\theta), 0)\} = \mathbb{E}\{\max(y^* - y, 0)\}
$$

Using a probabilistic model $p_M(y|\theta)$ of the objective function one may write

$$
EI(\theta, p_M(y|\theta)) = \int_{-\infty}^{\infty} \max(y^* - y, 0)p_M(y|\theta)dy = \int_{0}^{y^*} (y^* - y)p_M(y|\theta)dy.
$$

The lower limit of the integral can be set to zero since the score $y$ is non-negative. Since a Gaussian is completely defined by its mean and variance there exists an analytic expression for Gaussian Processes specifically

$$
EI(\theta) = \begin{cases} 
(\mu(\theta) - f(\theta^*))\Phi\left(\frac{\mu(\theta) - f(\theta^*)}{\sigma(\theta)}\right) + \sigma(\theta)\phi\left(\frac{\mu(\theta) - f(\theta^*)}{\sigma(\theta)}\right), & \sigma(\theta) > 0 \\
0, & \sigma(\theta) = 0
\end{cases}
$$

where $\Phi$ is the cumulative distribution function and $\phi$ the probability density function of the standard normal distribution [11]. Again, $\theta^*$ is the best parameter set so far minimizing $f$. The symbols $\mu$ and $\sigma$ are the mean and variance of $y$, i.e. our best guess of $y$ and with what certainty, given our modeled surrogate function $p_M(y|\theta)$. For the special case of zero variance, which can be chosen for evaluations of the actual black-box function, the expected improvement is set to zero since no improvement is possible when the value is known with certainty.
2.4.4 Designing the acquisition function

Acquisition functions usually have several design parameters that can be tuned. Since the aim is to implement an auto-tuner it is preferred to avoid the need of manually tuning the parameters of the auto-tuner itself. A common approach to automatically tuning the acquisition function is to choose the hyperparameter set $\alpha$ that maximizes the likelihood of the gathered data from the initialization phase. If the model is initiated by evaluating $N$ different inputs $x$ the hyperparameter set of maximum likelihood is given by

$$\alpha_{ML} = \arg \max_{\alpha} p_M(y|x_n^{N}_{n=1}, \alpha), \quad y = [y_1, ..., y_N]^{T}. \quad (27)$$

Another approach, that has been shown to improve the convergence of SMBO, is to integrate over these hyperparameters using a Markov Chain Monte Carlo (MCMC) algorithm. This method is referred to as Monte Carlo Acquisition

$$MCA(x; \{x_n, y_n\}_{n=1}^{N}) = \int a(x; \{x_n, y_n\}_{n=1}^{N}, \alpha)p_M(\alpha|x_n, y_n) d^{m}\alpha$$

or

$$MCA(x) = \int a(x; \alpha)p_M(\alpha) d^{m}\alpha, \quad m = \text{dim}(\alpha) \quad (28)$$

where $a$ could be any acquisition function and $\alpha$ are the hyperparameters defining that acquisition function. The $x$ and $y$ denote the inputs and the output of our black box function. In our case $x$ would be the control parameters $\theta$ that are to be optimized by the SMBO. The $N$ observations of $x$ and $y$ can also be viewed as hyperparameters defining the acquisition function through the GP, but they may be omitted for clarity. This integral can be estimated through Monte Carlo sampling. The result is a more general blend of acquisition functions where the hyperparameters that fit the data the best are dominating, but all other hyperparameters are considered to an extent proportional to their importance. This is a way of avoiding overfitting the model [15].

2.4.5 Maximizing the acquisition function

Choosing an efficient optimizer for the acquisition function is of little importance since it demands far less computing power than evaluating the true black box function. It is, however, important that the optimizer finds the global maximum, not only a local, to keep the number of iterations to a minimum. The hyperparameters of the acquisition function are commonly the length scale in every dimension, covariance amplitude, the constant mean of the GP and its noise level.
3 Method

The aim is to automate the process of tuning the parameters of a mobile robots motion controller. This process should not only have the ability to find well performing sets of parameters but do so efficiently. Hence several approaches to auto-tuning will be investigated and evaluated. What is considered well performing is defined in section 3.3.4 where the characteristics described, such as positional error, should be minimized. The efficiency of each automated process will be measured by the convergence as per definition 3.3. First, an explanation of the choice of optimizers to investigate is provided, then the package of optimizers used is presented and finally the methods of evaluating the performance.

3.1 The Choice of Optimizers to Investigate

The optimizers investigated in this work are grid search, random search, LHS search, see section 1.2, and Sequential Model-based Bayesian Optimizer (SMBO), see section 2.2.1. In turn, the SMBO can be designed with different probabilistic models, acquisition functions and covariance kernels. See sections 2.3 and B, 2.4 and 2.3.3 respectively.

The method used for auto-tuning has to comply with the following requirements for this application. The parameter sets suggested by the auto-tuner need to be evaluated running the real warehouse robot or a simulated run. It needs to be general enough to handle any type of parameters and versions of the robot for high usability. The method has to be data efficient, minimizing the number of evaluations as far as possible.

The requirement of evaluating the performance using simulation or real runs prohibits the derivation of an analytic mapping between parameter sets and performance score. It is only possible to evaluate the so called utility-, performance- or objective function by sampling. Only the input-output pairs can be made available. It is, per definition, a black box function. The art of black box optimization often goes hand in hand with derivative free optimization [23, 24]. This is due to the fact that black box functions are often complex and therefore expensive to evaluate. The great cost of evaluation limits the number of samples which in turn makes it difficult to approximate the derivative. Black box functions, as in this simulation and real run application, are often very noisy further complicating the calculation of the derivative since small deviation greatly impacts the derivative when data points are scarce.

There are plenty of derivative-free optimizers, such as direct search algorithms, but the most important and prominent are the model-based optimizers [25]. It may also be referred to as surrogate-based optimization since the model is used as a surrogate for the black box function. Forrester and Kaine advocates the use of such optimizers for tuning parameters using heavy simulations [26]. They present recent (2009) advances within the field such as the use of Gaussian processes and Bayesian selection techniques. One such technique is SMBO.

Many researchers have shown that SMBO is an efficient approach to optimizing expensive black box functions [11, 13]. Recall section 2.2.1 explaining the
method. It is considered the prominent learning-based optimizer and is the current research focus [27]. In contrast to gradient methods, Bayesian optimization is a derivative free approach and can be given samples located anywhere in the parameter space [12]. This means that, although SMBO does so automatically and sequentially, any samples can be chosen when building the probabilistic model.

To provide a fair setting when evaluating the different probabilistic models and acquisition functions Matérn covariance kernel is used constantly. See table 1 for the general expression. More specifically, Matérn with $\nu = \frac{5}{2}$ is used and can be simplified to

$$K(x_p, x_q) = \left(1 + \sqrt{\frac{5r}{l}} + \frac{5r^2}{3l^2}\right) \exp\left(-\sqrt{\frac{5r}{l}}\right), \quad r = |x_p - x_q|. \quad (29)$$

Bergstra and Bengio have studied and compared manual-, random- and grid search [28]. They acknowledge grid search and manual search, or a combination of the two, as the dominant methods of hyperparameter optimization even after the introduction of SMBO by Hutter in 2009 [29] among other advances. The reason for this, they argue, comes down to a few circumstances. Manual search gives the researchers some insight into the affect different parameters have on the result and does not demand an optimizer to be investigated, developed, bought or implemented. Depending on the accuracy demanded, the time available, the size of the parameter space etc. this might be too big of a threshold to overcome. The simplest optimizer, thus having the smallest threshold, is grid search which is simple to implement and parallelize since the order of samples does not matter. They also conclude that random search has the additional flexibility of making up a complete experiment using any subset of samples. The sampling can stop at any time, failed samples can be removed and any samples drawn with the same distribution can be added without affecting the validity of the experiment.

### 3.2 Implementation using GPyOpt Package

The auto-tuner is implemented as a Python script with the GPyOpt package imported. The package is implemented by the research group in machine learning at the University of Sheffield and made available on GitHub [30]. It is user friendly and has straight forward functions with input arguments for every design choice of the SMBO mentioned in this report. The objective function written handles all the communication with the simulator. When the SMBO requests an evaluation of a specific parameter set a json file is written containing all the necessary information the simulator needs. A system call is then made to run the simulation using the provided json file. The SMBO awaits the completion of the simulated run and then reads a csv file produced by the simulator containing recordings of the quantities necessary for the calculation of the objective score. Ten such simulation and resulting csv files are needed to calculate the objective score of the corresponding parameter set. What the physical quantities measured are, and how the objective function is calculated, is described in section 3.3.4. The standard grid search algorithm was implemented without the use of the GPyOpt package.
3.3 Test Performance in Simulator

The simulator is based on PyBullet, which is a Python module with bindings for Bullet. Bullet is a physics engine with capabilities of simulating both soft and rigid body dynamics. It may also be used for collision detection, forward and inverse dynamics and kinematics as well as ray intersection queries. Many different file format can be used to load articulated bodies into the simulator. Bullet is written in C++ and licensed under Zlib. The code is open sourced and free for commercial use on any platform. The source code is hosted on GitHub together with documentation [31, 32].

To facilitate comparisons between the design choices of the SMBO a clear, unambiguous, definition of convergence is needed. **Convergence in the context of this experimental setup** is reached when the score is 90% better than the average score compared to the best score ever found. The average and best score are both obtained by an exhaustive grid search of defined resolution and boundaries.

The regret of a function $f$ evaluated with parameters $\{\theta_k\}_{k=1}^N$ is determined by

$$R(f, \{\theta_k\}_{k=1}^N) = \sum_{i=1}^N (f(\theta_i) - f(\theta^*)),$$

where the benchmark $\theta^*$ is the best parameter set ever found, but not necessarily the true optimal. Although each iteration of the SMBO may result in a greater value of the objective function only the best score so far is kept resulting in a monotonically decreasing convergence function. This avoids the issue of considering functions that reach the defined convergence interval but later divergences from it, at least temporarily. The convergence curve might however exhibit more properties of interest such as how much it continues to improve after the 90% improvement has been reached. This property has strong connections to the so called exploration-exploitation trade-off. Exploiting the data results in quickly finding a local maximum, which might be sufficient to reach the static convergence definition, but with no exploration at all the global maximum can only rarely be found by initial samples that happen to be favorable. The initial samples might just as well be inferior if the objective function truly is a black-box function which, by definition, implies that nothing can be assumed of where the global maximum is when initializing the model. To better include the exploration capabilities of the differently designed optimizers in the result section the final optimal value after a given number of iterations will also be included.

3.3.1 Different boundaries

Two different set of boundaries are investigated. The first one, that is used as standard unless otherwise stated, is broad in an effort to include any reasonable value. This is done to investigate how far the process can be automated. It is impossible to decide on limits without bringing any of your knowledge, but the limits are chosen making as few assumptions as possible that should work in more general cases without adapting the limits. The maximum and minimum value of every parameter is set to the same. The second set of boundaries is
based on expert knowledge and analysis. This case is less about automating and more about improving slightly on a high performing controller developed by an expert. The limits are set in comparison to the parameter values chosen by an expert. The lower limits are 50% of the chosen value for every parameter and the upper limits are 150% of those.

3.3.2 Different probabilistic models

The different probabilistic models investigated are GP, sparse GP and GP with Monte Carlo Markov Chain integration of the acquisition function. Random forest and warped GP could not be tested because of implementation error in the GPyOpt package.

3.3.3 Different acquisition functions

One may not only choose different types of acquisition functions, but the hyperparameters of each type are also a part of the design. However, this process is automated in this study by maximizing the likelihood of the initial data as was mentioned as a possible solution in section 2.4.4.

3.3.4 Defining the objective function

The process of finding a good objective function is a central part of designing machine learning algorithms such as SMBO. In order to get the desired results, what is desired must be well defined by the objective function. In the case studied the score is very volatile due to intrinsic randomness of the simulation. The same input can give widely differing output scores between evaluations because a new simulation is run each time. To measure the magnitude of this noise several evaluations of the same input were made. Two such series of evaluations of different inputs can be seen in Figure 10. The SMBO is capable of modeling noise in the output score, but the noise can also be drastically reduced by averaging over several simulation runs. Less noise helps the SMBO build its probabilistic model. Without this averaging it is very likely that the SMBO ends up with a very unstable parameter set that just happened to output the optimal score during a run. One might ask: is a parameter set really optimal just because it happened to give good performance once? In this application the answer is no since the robot should be robust to disturbances. However, the improvement in the stability of the results when running multiple runs per evaluation comes at the cost of slower convergence. When running simulations this additional cost is warranted.

The qualities that are considered in the objective function are the following:

**Positional Error**

The main objective. The objective function should reflect the desire of high accuracy of the controller, but it should not come at too high a cost. The remaining physical quantities secondary, used to prevent parameters that may give high accuracy sometimes but are unstable and aggressive. The importance of the accuracy is the highest at the beginning and end of the run since the first grid cells should be made vacant as soon as possible and overshooting the last
cell could result in a crash if another robot was treating it as vacant. However, it was found during the implementation phase that keeping a uniform importance during the entire run produced stable parameter sets with high accuracy at the start and end of the run as well.

**Signal currents**

It is preferred to improve the accuracy of the controller without suffering increased wear and tear as well as energy use. By minimizing the signal current magnitude the robot should drive smoother and hence more efficient; high acceleration and braking involves more energy loss.

**Discrete derivative of signal current**

Two signal currents can have the same standard deviation but one switching between positive and negative deviation a lot more often than the other. Rapid fluctuations in signal current causes increased wear and energy consumption much like high current magnitudes. The current will have to vary a lot to quickly react to different disturbances, but many parameter sets give the unwanted effect of jumping between the upper and lower limit of the current. In order to focus the penalty on those extreme spikes in the derivative the score is also squared.

**Error in wheel speed**

Finally, it is desirable to avoid slippage. The slippage is already reduced due to the minimization of signal currents and its derivative, but a wheel may slip a lot even at low current. This happens if the wheel loses grip. Not only should the current be low in this situation, but the wheel speed should not increase considerably either. All wheel speeds are compared separately to the desired speed of the robot.

All qualities are measured as the standard deviation of the actual value compared to the desired value at each time step. Additionally, for the derivative of the currents, the standard deviation for each wheel is squared.

**Stability**

Due to intrinsic randomness in the simulation all these scores may vary a lot between runs. To avoid much of this uncertainty each score is averaged over several simulations, ten to be specific, and to penalize unstable parameter sets the standard deviation of the scores over all simulations is added. The final score is hence a sum of four mean values and four standard deviation, all with different weights.

### 3.4 Comparing SMBO to other Tuning Approaches

This is all done in the simulator using the same package GPyOpt except for the standard grid search implemented from scratch.

#### 3.4.1 Standard grid search and Latin Hypercube sampling

By only running the initialization phase of the SMBO it is possible to perform a kind of grid search. This is done by setting the initial design type to latin.
Latin hypercube sampling is based on an extension of the Latin square, which is a matrix where each sample appears only once in its row and column [33]. The algorithm guarantees that all grid rows in every dimension are sampled. Given the number of samples requested each dimension, or parameter in this case, will be partitioned in the same number of intervals. The value in the middle of each interval will then be combined with the other parameters in a random fashion apart from the fact that every value of each parameter may only appear once. See figure 9 for an example 10 samples and two parameters $x$ and $y$. The sampling is quasi-random, with replacement and constrained to the centers of the grid cells.

![Figure 9: Latin hypercube sampling](image)

The Latin hypercube sampling procedure combines advantages of both classic grid search and random search. The sampling is exhaustive in the sense that it guarantees $n$ different, equidistant, values of each parameter sampled, where $n$ is the number of samples. This maximizes the spread of samples in each dimension. However, it does not sample all possible combinations. The randomness in the combinations chosen helps decorrelate the parameters. Standard grid search was also implemented and used.

### 3.4.2 Random search

Similarly to grid search the initial design type was set to random and the optimizer is set to finish after the initialization phase, before the first model is even built. See section 1.2 for more on random search.

### 3.4.3 Manual tuning

A single set of parameters derived by a human expert is evaluated. The derivation is mainly theoretical but includes some iterative tuning based on analyzed responses.
4 Results

4.1 Volatility of Scores for Same Input

As can be seen in Figure 10 some parameter sets are very unstable compared to others. After averaging over ten simulation for every evaluation of the score it becomes more stable as seen in Figure 11. The score also includes a penalty term for instability as explained in the method section 3.3.4. This preprocessing is part of the definition of the objective score used the entirety of this report.

Figure 10: A comparison between two different input parameter sets, one where the evaluated score differs widely between simulation runs and another more stable.
Figure 11: Evaluation of the same two parameter sets as in Figure 10 but with stabilized scores. The variation in the scores is reduced by averaging the result of 10 simulations for every evaluation of the objective function.

4.2 Different Probabilistic Models

The probabilistic models GP, sparse GP and GP with Markov Chain Monte Carlo acquisition are compared in Figure 12. The same acquisition function, namely expected improvement, is used in all cases. The result is produced by running each variant of the SMBO twice with one hundred iterations and then averaging.
4.3 Different Acquisition Functions

The choice of acquisition function is evaluated the same way as were the probabilistic models, see Figure 13. The acquisition functions tested are expected improvement, probability of improvement and lower confidence bound. The probabilistic model used is standard GP in every case.
4.4 Different Boundaries

Figure 14 shows the standard GP with expected improvement as acquisition function optimizing in the vicinity of the parameter set chosen by an expert. The optimizer had a lower bound of 50% below and 150% above the values chosen by the expert for each parameter.
Auto-tuning bounded near expert’s choice of parameters

Figure 14: Comparison between the score of the parameter set chosen by a human expert and the score of an SMBO searching in a tight feature space around the same parameter set. The space was limited to lower bounds of 50% and upper bounds of 150% of every parameter value chosen by the expert.
Figure 15: Every design choice tested in one graph for comparison. The best score ever found is in the context of the large feature space used. It was found running GP EI 1000 iterations, see Figure 16. The 90% improvement is shown according to definition 3.3.
<table>
<thead>
<tr>
<th>Model</th>
<th>Regret</th>
<th>Time /iteration</th>
<th>Total time</th>
<th>Best Score Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>GP EI</td>
<td>8.99</td>
<td>52 s</td>
<td>1.4131 h</td>
<td>0.1104</td>
</tr>
<tr>
<td>GP MPI</td>
<td>10.83</td>
<td>57 s</td>
<td>1.5807 h</td>
<td>0.1366</td>
</tr>
<tr>
<td>GP LCB</td>
<td>9.50</td>
<td>51 s</td>
<td>1.4292 h</td>
<td>0.1177</td>
</tr>
<tr>
<td>Sparse GP EI</td>
<td>11.41</td>
<td>52 s</td>
<td>1.43 h</td>
<td>0.1254</td>
</tr>
<tr>
<td>Sparse GP MPI</td>
<td>11.10</td>
<td>52 s</td>
<td>1.43 h</td>
<td>0.1274</td>
</tr>
<tr>
<td>Sparse GP LCB</td>
<td>8.95</td>
<td>52 s</td>
<td>1.44 h</td>
<td>0.1150</td>
</tr>
<tr>
<td>GP MCMC EI</td>
<td>10.38</td>
<td>164 s</td>
<td>4.56 h</td>
<td>0.1161</td>
</tr>
<tr>
<td>GP MCMC MPI</td>
<td>11.07</td>
<td>127 s</td>
<td>3.53 h</td>
<td>0.1363</td>
</tr>
<tr>
<td>GP MCMC LCB</td>
<td>10.09</td>
<td>129 s</td>
<td>3.59 h</td>
<td>0.1130</td>
</tr>
<tr>
<td>LHS Grid Search</td>
<td>10.52</td>
<td>50 s</td>
<td>1.40 h</td>
<td>0.1225</td>
</tr>
<tr>
<td>Random</td>
<td>9.56</td>
<td>50 s</td>
<td>1.41 h</td>
<td>0.1314</td>
</tr>
<tr>
<td>LHS Grid Search 1000 iterations</td>
<td>10.52*</td>
<td>51 s</td>
<td>14.11 h</td>
<td>0.1225**</td>
</tr>
<tr>
<td>Standard Grid Search 2048 iter.</td>
<td>10.04*</td>
<td>50 s</td>
<td>28.66 h</td>
<td>0.1229**</td>
</tr>
<tr>
<td>GP EI 1000 iter.</td>
<td>10.89*</td>
<td>65 s</td>
<td>18.13 h</td>
<td>0.0502**</td>
</tr>
</tbody>
</table>

Table 2: Convergence for Differently Designed SMBOs. All designs, but the three last, are averaged over two runs of the optimizer. The star * indicates that the regret is calculated based on the first 100 iterations, for a valid comparison, but these methods continue minimizing the score for many more iterations. The double star ** stresses that the Best Score Found for these optimizers were found after many more iterations than all other optimizers.

4.5 Performance of Different Optimizers

A summary of all differently designed variants of SMBO is shown in Figure 15 together with some additional values for comparison. Those are from the convergence of a grid search with ten times more iterations than the SMBOs, its final and average values together with a line showing 90% improvement from average to final value. The lowest line shows the best value ever found obtained by running GP EI one thousand iterations in the vicinity of the parameter set chosen by an expert shown in Figure 14. Table 2 shows the convergence measured as regret for simpler comparison together with the best score found for each design choice and corresponding computation times.
Figure 16: GP EI compared to grid search with two different resolutions, or fineness.

The GP with EI as acquisition function is compared to grid search in Figure 16. Two different numbers of iterations are shown since it causes the LHS to have different convergence behaviour because of the different implied resolutions.
5 Discussion

The incredibly high volatility in the score, for some of the parameter inputs, seen in Figure 10 was found to be a big concern. It can even be seen that these two parameter sets, producing very different control behaviours, sometimes get the same objective performance score. That would result in a flat probabilistic model giving no indication on what parameter set to evaluate next. After the preprocessing of averaging and adding a penalty for instability the difference between the scores for the two parameter sets was always at least 0.5, see Figure 11. In other words the unstable parameter sets’ score was at least 100% higher, eradicating this issue.

Figure 12 shows no significant difference between the choices of probabilistic model. They are all, however, based on Gaussian processes. The difference would probably be greater if random forest, which is fundamentally different, could be tested. See appendix B. The standard GP performed best, but it could very well be because of chance. The same reasoning goes with the choice of acquisition function shown in Figure 13, where EI performed best with a tiny margin. With a bit higher certainty it can be said that LCB acquisition was the worst performing acquisition function.

When guiding the auto-tuner to a feature space region near the manually chosen parameter values it showed great potential. See Figure 14. If restricted to 100 iterations the LHS outperformed the other, probably because it quickly sweeps through the feature space without getting distracted by local minima. LHS with 1000 iterations has ten times higher resolution at the cost of 10 times more evaluations. This explains why it converges slower but finds a better score at the end compared to LHS with 100 iterations. The SMBO with GP EI setting found a slightly better score, but the learning did not seem to have a substantial contribution. In the summarizing Figure 15 all optimizers seem to quickly plateau around the score 0.12, but with very different parameter values. Again, this supports the assertion that the objective score has a complex surface with many local minima confusing learning based methods. The exploration seems to be of high importance since the random based optimizers LHS and random search almost perform on par with the learning based SMBOs. More evidence supporting the occurrence of this plateau is available in Figure 16. Here the SMBO outperforms LHS in both setting; 100 and 1000 iterations. After about 900 iterations the SMBO found significantly better performing parameters, on par with the manually chosen parameters.

Much time was spent on defining the objective function to be minimized. Similarities can be drawn to defining the input signals for motion controllers; what attribute do we want to minimize? A position controller, for example, can be given the positional error as input, and the same positional error can then be used in the objective function to evaluate the performance of the controller. The objective score answers the question; did the auto-tuner make the controller minimize what it was supposed to minimize?

One interesting aspect of SMBO is that even with a local optimizer for the acquisition function it should eventually find the global maximum of the objec-
tive function since all acquisition functions are explorative to some degree.

It can be argued that a true auto-tuner should tune the parameters completely automatically. It should be enough to specify the parameters to be tuned and the objective function. The objective function defines the objective of the tuning and depends on the application. No set of parameters is optimal in every conceivable way. But apart from that there should be no design choices needed. A major part of this paper has been on investigating design choices affect on the performance of the SMBO for this application but automating this process fully would be beneficial and time saving. Every design choice made in this paper could in theory be automated, but if it can clearly be shown in which kind of applications one design outperforms all other then choosing those setting could speed up the automated process considerably. Unfortunately, no substantial performance difference between the different designs of the SMBO was found in this work other than MCMC integration consuming a lot more computational time without improving the objective score.

Another big issue was setting the boundaries. Auto-tuning should preferably not require expertise and analysis to get it running, but how should the boundaries of the parameter space investigated be set? The simplest boundaries to set are probably at the physical limits of the system which should only require knowledge about the power supply and similar. One way to automate this process could be to set small boundaries and increase the boundary incrementally if the optimal parameter is found right on the boundary, but what if the initial tight boundaries are in a region that is completely off? The SMBO demands boundaries to be chosen, and there is probably no way to automate this.

Designing the objective function also comes down to data efficiency. If the function gives a detailed picture of the control performance a well performing parameter set should be found quicker. An effort was made to include as much of the available and valuable simulation data as possible for each run. There is however the risk of diverting the focus to qualities that are of less importance than other if the objective function is too complex.

Comparing Latin hypercube sampling with standard grid search it does not provide a model that can be analysed one parameter at a time, keeping the other constant. This is because, by definition, the hypercube is sampled with all other parameter values changing whenever you change one. It is not possible to learn how the objective function changes with one parameter in this setting. Instead LHS provides a model of how changing one parameter changes the score of the objective function in general, randomizing over the other parameters averaging their influence. As can be seen in figure 1 LHS and random sampling are similar in the sense that more values of each parameter are sampled compared to standard grid search increasing the opportunity of finding that a single parameter has the major influence on the score. This is of course at the cost of reduced chance of finding patterns in how parameters may work in tandem, affecting the score substantially when synced in some way.

Sparse GP would probably be considerably faster than GP if the number of iterations were a lot higher since the models would be constructed using many
more data point quickly growing the difference in computational cost of GP
compared to sparse GP. This is because the cost grows with a power of 3 for
standard GP but 2 for sparse GP.
6 Conclusion

An auto-tuner was developed to auto-tune the parameters of the motion controller of a mobile robot. This included the use of the package GPyOpt available online. The SMBO implemented in the package was applied to this purpose by letting it request evaluations of specific parameter sets in simulated runs. An objective function was developed to define how well a parameter set performs based on recorded data from the simulation.

Based on the objective function defined, the auto-tuner performs on par with manual tuning after less than 1000 iterations when optimizing over a large feature space with slack boundaries. This auto-tuning process takes about 18 hours running on a standard CPU. Allowing more iterations would outperform manual tuning but the convergence rate is unknown since it was not investigated further. However, greatly reducing the feature space to the best parameter values found by manual tuning plus and minus 50% showed that much better performing parameter values could be found in the neighbourhood based on the definition of the objective function. No more than 5 iterations were ever needed.

Integration over the parameters of the acquisition function using the Markov Chain Monte Carlo algorithm was found not to be suitable for this application. The process more than doubled the computational time needed compared to the other designs without improving the convergence rate.

The results indicate that the GP design with expected improvement acquisition could be the optimal design for this application. Sparse GP did not reduce the computational time, as it should, but it probably does in other applications with much higher numbers of training samples. Overall the difference between different optimizers at 100 iterations could be considered negligible since randomness in the simulation and initialization can have an effect of similar magnitude. When running 1000 iteration, however, the difference in best score found was large between Latin hypercube sampling and SMBO. The objective score, to be minimized, was 60% lower for the SMBO compared to Latin hypercube sampling, but more runs are needed to rule out the influence of noise. What can be said, with high confidence, is that standard grid search had the worst performance of all auto-tuners in this setting. The algorithm was run with the smallest uniform grid possible for an 11-dimensional space, namely 2048 iteration, which is more than 20 times the number of evaluations needed for the other auto-tuners but without improving the objective score.
References


37


A Warped Gaussian process regression

The standard GP model is often a good approximation of real world phenomena, but many distribution are still far from Gaussian. By preprocessing the data the GP framework can be generalized to handle non-Gaussian processes and non-Gaussian noise. By introducing a latent variable, call it $z$, which is modelled by a GP

$$- \log p(z|x) = \frac{1}{2} (z^T (K + \sigma_n^2 I)^{-1} z + \log |K + \sigma_n^2 I| + n \log 2\pi)$$  \hspace{1cm} (30)$$

it can bridge the gap between non-Gaussian data and the GP regression. This is done by transforming the scores into a latent space, warping the output space of the black-box function

$$z_n = f(y_n; \Psi) \quad \forall n, \hspace{1cm} (31)$$

where $f(\cdot)$ is a non-linear monotonic function mapping the range of real numbers. Without this requirement the probability measure would not be conserved. The function is parameterized by $\Psi$ which can be tuned by maximizing the likelihood of the data. Using this mapping, the probabilistic model of the black box function is given by

$$- \log p(y|x, \Psi) =$$

$$= \frac{1}{2} (f(y)^T (K + \sigma_n^2 I)^{-1} f(y) + \log |K + \sigma_n^2 I| + n \log 2\pi) +$$

$$+ \sum_{n=1}^{N} \log \left| \frac{\partial f(t)}{\partial t} \right|_{t=t_n}, \hspace{1cm} (32)$$

where the last term originates from Jacobian of the variable substitution. Snelson et al. [34] used a combination of hyperbolic functions for their monotonically increasing yet complex shape, and a linear function to fulfill the requirement of mapping to the entire set of real numbers [34].

$$f(y_n; \Psi) = y_n + \sum_{k=1}^{K} a_k \tanh(b_k (y_n + c_k)), \quad a_k, b_k \geq 0, \quad \Psi = \{a, b, c\} \hspace{1cm} (33)$$
Figure 17: Regression results produced by Edward Snelson et al. showing the true distribution generating the data in dotted lines, a GP’s prediction in dashed lines and a warped GP’s prediction in solid lines. For each triplet of lines in (a) the center line represents the median and outer lines the $2\sigma$ percentiles. The predictive probability densities at $-\pi/4$ are shown in (b) corresponding to a cross section at the grey line in (a) [34].

The improved flexibility and better fit to the data, compared to standard GP, is demonstrated in Figure 17.

B Random Forest

Random Forest, or Random Decision Forest, is a machine learning method for classification and regression based on an ensemble of Decision Trees. In its simplest form a number of decision trees are trained each using a randomly selected subspace of the feature space [35]. The regression or classification of unseen data is then done by combining the classification of the individual trees into one by using a discriminant function. The discriminant function can be simply the average probability of the input belonging to each class over every tree. The input is then classified by choosing the class for which the discriminant function is maximized.

Today, random forest is typically combined with bootstrap aggregating, or bagging, which is also a method of averaging over predictors to form a new predictor of higher accuracy [36]. The bagging predictor is defined as

$$y_{\text{bagging}}(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} y_m(\mathbf{x}),$$

(34)

where each $y_m$ is a predictor trained on an individual data set, each generated from randomly sampling the available data with replacement. The method of generating more data without doing further evaluations is called bootstrapping. The randomness introduced by generating bootstrap data and randomly choosing feature subspaces helps avoiding overfitting.