Learning Occlusions in Robotic Systems: How to Prevent Robots from Hiding Themselves

Jakob Nazarenus[1][a], Simon Reichhuber[2][b], Manuel Amersdorfer[3][c], Lukas Elsner[4][d], Reinhard Koch[1][e], Sven Tomforde[2][f] and Hossam Abbas[4][g]

[1]*Multimedia Information Processing Group, Kiel University, Hermann-Rodewald-Str. 3, 24118 Kiel, Germany*

[2]*Intelligent Systems, Kiel University, Germany, Hermann-Rodewald-Str. 3, 24118 Kiel, Germany*

[3]*Digital Process Engineering Group, Karlsruhe Institute of Technology, Hertzstr. 16, 76187 Karlsruhe, Germany*

[4]*Chair of Automation and Control, Kiel University, Kaiserstr. 2, 24143 Kiel, Germany*

Keywords: Vision and Perception, Robot and Multi-Robot Systems, Simulation, Neural Networks, Classification, Autonomous Systems.

Abstract: In many applications, robotic systems are monitored via camera systems. This helps with monitoring automated production processes, anomaly detection, and the refinement of the estimated robot's pose via optical tracking systems. While providing high precision and flexibility, the main limitation of such systems is their line-of-sight constraint. In this paper, we propose a lightweight solution for automatically learning this occluded space to provide continuously observable robot trajectories. This is achieved by an initial autonomous calibration procedure and subsequent training of a simple neural network. During operation, this network provides a prediction of the visibility status with a balanced accuracy of 90% as well as a gradient that leads the robot to a more well-observed area. The prediction and gradient computations run with sub-ms latency and allow for modular integration into existing dynamic trajectory-planning algorithms to ensure high visibility of the desired target.

1 INTRODUCTION

With increasing computing capability, it is possible to make robotic systems more and more intelligent. This includes improved perception of the environment and the robotic state so that an appropriate action can be calculated based on this. Examples such as *Boston Dynamics*[1] show that robots can solve path-finding problems even in unfamiliar terrain in a similar way to animals and can hardly be distinguished from real animals in their movement sequences (Guizzo, 2019). The issue of localization of robotic systems, especially multi-joint robotic arms, is challenging for robotics and sensor technology. Various approaches directly place sensors on board to detect the near proximity around the moving manipulator and use known structures of the environment for localization (Fan et al., 2021). Because of the speed that can be achieved at the end of the arm, sensors must provide high frame rates by simultaneously lowering the resolution for real-time processing. In contrast to the on-board approach, external, stationary cameras exploit well-known reference positions enabling highly accurate positioning of moving objects equipped with optical reference markers (Liu et al., 2020). Besides the local restriction within the range of the camera system, there is another crucial drawback in such systems, namely the line-of-sight occlusions. With the term line-of-sight occlusion or simply occlusion we describe points within the viewing cone of a camera where the direct line of sight to the camera is interrupted by an obstacle, which means the obstacle itself also counts as part of the occlusion (cf. Figure 1). In the simple case, we have the exact position of the robot and the environment, which allows us

[a] https://orcid.org/0000-0002-6800-2462
[b] https://orcid.org/0000-0001-8951-8962
[c] https://orcid.org/0000-0002-6416-3453
[d] https://orcid.org/0009-0001-8097-2373
[e] https://orcid.org/0000-0003-4398-1569
[f] https://orcid.org/0000-0002-5825-8915
[g] https://orcid.org/0000-0002-5264-5906
[1]https://bostondynamics.com/ (accessed January 29, 2024)

Nazarenus, J., Reichhuber, S., Amersdorfer, M., Elsner, L., Koch, R., Tomforde, S. and Abbas, H.
Learning Occlusions in Robotic Systems: How to Prevent Robots from Hiding Themselves.
DOI: 10.5220/0012431000003636
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 2*, pages 482-492
ISBN: 978-989-758-680-4; ISSN: 2184-433X
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

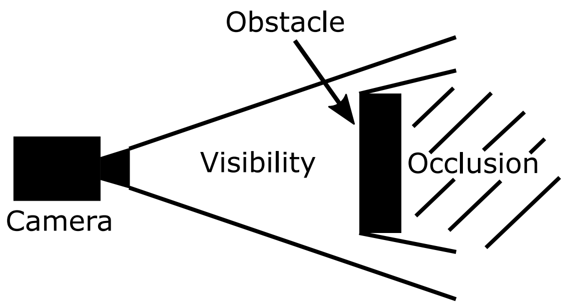


Figure 1: Terminology line-of-sight occlusion.

to calculate the occlusion geometrically. However, it is possible that the surroundings are unknown or that the robot obscures itself. Especially in applications where robotic arm movements are tracked by optical markers, there is not only an interruption of the line-of-sight when an external object interferes but also self-occlusions occur at certain joint configurations. When the reference point on the robot is occluded, the pose estimation falls back to the imprecise forward kinematics of the robot causing severe accuracy issues.

To anticipate and prevent occluded states, we suggest to construct a binary map of occluded robot states given by a learned approximation, which can be used for the trajectory calculation. The latter requires training data that represents the robot state space and its occlusions fully sampling the possible robot states. For this purpose, we provide a sampling routine that can be used to provide data to model the occlusions. At the end, we discuss the possibility of using the occlusion model to refine path planning by minimizing the duration of occluded optical references. For demonstration, we provide a showcase where a robotic arm holds its end position statically while simultaneously moving the rest of the arm to reduce the occlusions.

The remainder of the paper is structured as follows: The subsequent Section 2 lists the current state of the literature about occlusions and robotic arm localization. Section 3 is divided into three parts: First, we introduce the problem of learning occlusions as a binary classification problem and discuss the methodology of how to generate trajectories for the training, second, we describe simulation methods and the real-world experimental setup, and third, we introduce the deep learning model for classification. Afterward, in Section 4, we present all results visually and discuss them in Section 5. Finally, in Section 6, we conclude the work and point out some further ideas for future work.

2 RELATED WORK

The issue of detecting occlusions of important areas that occur when recording robotic systems with static camera systems is highly task-dependent. It is often more appropriate to use a moving camera (Silva and Santos-Victor, 2001) and to analyze the occlusion using different perspectives. An explicit detection of occlusion in a stream-based fashion based on optical flows has already been shown (Ayvaci et al., 2012). Here, the obstacle or the camera is moving, which simplifies the detection of obstacles, since the line-of-sight between both is temporally given. However, this method is not applicable to static obstacles. Besides explicit detection of occlusions, there are two implicit methods of how to cope with them. First, the addition of stationary-placed sensors allows *looking behind* the occlusion. Instead of learning the spatial distribution of blind spots, some approaches try to reveal blind spots by merging other more pervasive sensors with the occluded sensor. A common way is the usage of Radio-Frequency sensors, which enable localization with centimeter-scale precision (Boroushaki et al., 2021). A radio-frequency perception is able to penetrate non-conducting materials enabling the tracking of an external Radio-Frequency Identification (RFID) reference.

Second, onboard sensors can measure the near proximity of all moving parts of the robot which turns the environment into the state of reference and obviously eliminates the line-of-sight from the reference (environment) to the robot. When certain known objects in the environment are detected, a relative localization is allowed. Knowing only the type and shape of the object, the robot is able to interact in-place with the object. Otherwise, if only the distance to collision is measured only collision avoidance is possible. Directly tracking the object of observation and its near proximity by onboard sensors is another way to prevent occlusions and collisions. For example, the tracking of human arm motions using an Inertial Measurement Unit (IMU) and potentiometer has been proposed (Shintemirov et al., 2020). As a ground truth to evaluate the localization accuracy of a capsule equipped with IMU and a camera system for endoscopy, Vedaei and Wahid placed the capsule on a robotic arm and tested movements that naturally occur gastrointestinally (Vedaei and Wahid, 2021).

Another onboard sensor approach is proximity sensing (Gandhi and Cervera, 2003), which has been shown in (Fan et al., 2021) using surface waves. Besides this external sensors are used to track the robotic system or parts that it interacts with, e.g. part localization in grasping applications (Zheng et al., 2018).

To lower the complexity of occlusions, for some tasks, it is appropriate to model them in only two dimensions. For example, given the task of mechanical search, where a manipulator has to find objects that are occluded by other ones, the occlusions can be visualized by a 2-dimensional heat map (Danielczuk et al., 2020). In the following, we use the term *occlusion* as an abbreviation for interruptions in the line-of-sight of a camera to the optical reference point at the robot end effector. The occlusions are therefore camera-specific. Enriching the joint space of the robot with the binary information about the line-of-sight of a camera allows the planning of trajectories that minimize the duration of line-of-sight interruption between the camera and the end effectors.

3 METHODS

The visibility of a point at an n-Degrees of Freedom (DoF) robot can be modeled by a function $f : \mathbb{R}^d \rightarrow \{0,1\}$ that takes the joint angles of the robot as an input and has a binary output with 1 representing the visible, and 0 representing the hidden state. The partial visibility of a non-point-like extended object can be expressed by loosening the function definition to allow mapping to values in the interval $[0,1]$. This function is highly complex as it depends on the pose of the camera, the geometry of the robotic system and its surroundings, as well as its kinematics. Our approach is to learn this function in a supervised manner by automatically sampling the domain of the function and observing the corresponding output. We then use these samples to employ efficient data-driven methods to find an estimate for f. To evaluate the capabilities of the employed methods, we used the Balanced Accuracy (bAcc), e. g. found in (Brodersen et al., 2010). For a set of matched inputs, it is given as the arithmetic mean of Sensitivity (SEN) and Specificity (SPEC)

$$\mathrm{bACC} = \frac{1}{2}(SEN + SPC) = \frac{1}{2}\left(\frac{TP}{P} + \frac{TN}{N}\right). \quad (1)$$

Here, TP (TN) is the number of correctly positively (negatively) classified samples, while P and N are the number of positive and negative samples. The balanced accuracy is used here because the dataset is highly imbalanced in favor of the visible samples. Similarly to the F1 score (Rijsbergen, 1979), the bAcc is used in such situations to create a more balanced measure for the problem. We chose the balanced accuracy as it is easily interpretable and in comparison to the F1 score does not put more emphasis on the positive class.

3.1 Trajectory Generation

For the supervised training of our occlusion prediction model, we need to obtain a set of points in the state space of the robot with their corresponding occlusion status. This training set needs to be large enough to allow learning-based models to generalize their knowledge for independently sampled test data. As a simple strategy, a set of points is uniformly sampled from the state space $Q \subseteq \mathbb{R}^n$ for a robot with n joints. While this approach might work well with simulated robots, it does not provide a traversable path for real robots due to violating the velocity and acceleration constraints. Therefore, an automatic generation of training trajectories is required. In this section, we describe a method to generate a continuous path closely resembling the uniform sampling approach under the given kinematic constraints. Fourier series are commonly used to design excitation trajectories for parameter identification in robotics (Swevers et al., 1997b; Swevers et al., 1997a; Park, 2006; Stürz et al., 2017). We utilize the concept in the form of smooth random functions (Filip et al., 2019). Here, the trajectory of each joint $i \in \{1,\ldots,n\}$ is defined as

$$q_i(t) = \sum_{k=0}^{m}\left[a_{i,k}\cos\left(\frac{2\pi kt}{T}\right) + b_{i,k}\sin\left(\frac{2\pi kt}{T}\right)\right] \quad (2)$$

with $b_{i,0} = 0$, where $a_{i,k}$ and $b_{i,k}$ are the Fourier coefficients, m is the number of modes, and T denotes the trajectory period. We normalize the generated function values to unit variance. In the following, we consider the coefficients $a_{i,k}$ and $b_{i,k}$ to be sampled from $\mathcal{N}(0, 1/(2m+1)$. An alternative method of optimizing these coefficients is shown in Appendix A. The corresponding trajectory velocity $\dot{q}_i(t)$ and accelerations $\ddot{q}_i(t)$ are obtained directly from the Fourier series' time derivatives. The trajectories have to satisfy the kinematic and dynamic constraints of the robotic manipulator. Let $(q_{i,\min}, q_{i,\max})$ be the lower and upper bound for the angle of robot joint i. Its angular velocity and acceleration limits are denoted as $\dot{q}_{i,\max}$ and $\ddot{q}_{i,\max}$ respectively. This leads to the state space definition $Q := \{q \in \mathbb{R}^n \,|q_{i,\min} \leq q_i \leq q_{i,\max}, i \in \{1,\ldots,n\}\}$.

An exemplary path for a single joint with $m = 10$ and $T = 2$ is shown in Figure 2. While for any t, its function value is randomly distributed according to $\mathcal{N}(0,1)$ under the random choice of the coefficients a_k and b_k (Filip et al., 2019), for a single path the distribution deviates from $\mathcal{N}(0,1)$ due to the individual function values not being independent. This can be seen in Figure 2, where the trajectory for $m = 10$ and $T = 2$ is relatively smooth.

With increasing m, the random function becomes

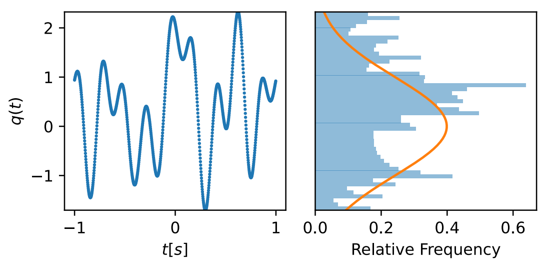


Figure 2: Smooth random path (left) with its spatial distribution (right) for a mode count of $m = 10$. The orange line shows the distribution for the limit of infinite modes or infinite paths.

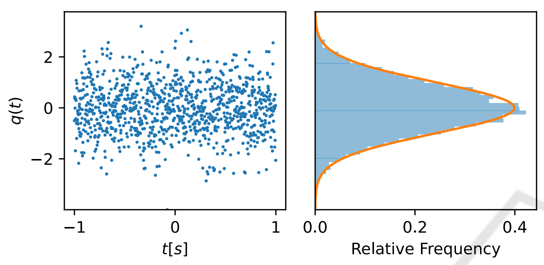


Figure 3: "Smooth" random path (left) with its spatial distribution (right) for a mode count of $m = 10^4$. The spatial distribution approaches its limit of a Gaussian distribution.

less smooth and oscillates more often. This reduces the dependency of successive function values and thus the deviation from $\mathcal{N}(0,1)$. This behavior can be seen in Figure 3 for $m = 10^4$ and $T = 2$.

While this randomness is desirable for the efficient sampling of the joint space, there are limits given by the maximum angular speed and acceleration of the robot. When observing the first and second derivatives, we see them increasing as shown in Figure 4. This prevents the arbitrary increase of the mode count m, as it would inevitably cause violations of the robot's kinematic constraints. In our case, we want to generate a path that maximizes the variability under the given constraints for the function values and its first two derivatives. For this purpose, we propose the following approach shown in Algorithm 1. For each

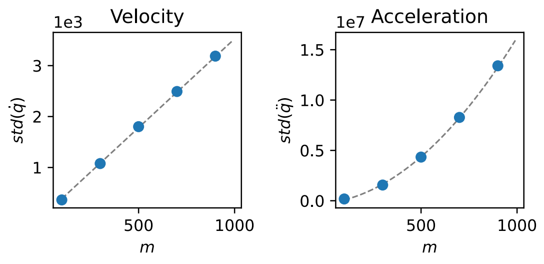


Figure 4: Increasing mode counts leads to higher velocities and accelerations. The dashed line shows a linear fit for the velocity and a quadratic fit for the acceleration.

joint, we initially find the highest mode count m that causes the $\dot{q}$ and $\ddot{q}$ constraints to not be violated most of the time. To achieve this, we choose a threshold of 95% (2σ) of the data to be below the given limits. Increasing this threshold causes fewer violations, but at the same time reduces the reached mode count m and thus the variability of the generated data. For the generated trajectories, we enforce the $\dot{q}$ and $\ddot{q}$ constraints by locally slowing down the time via the function `localTimeScaling()`. Then, any violations of the angular constraints are removed by replacing the violating parts of the trajectory with quadratic polynomials via the function `quadraticClipping()`. As both functions alter the overall duration of our trajectory, we enclose this process with a search for the optimal input duration and terminate once the search interval falls below a predefined threshold ΔT. This threshold causes the resulting trajectories for all joints to slightly differ in their duration, which necessitates the artificial slowing of all but the longest generated trajectory via the method `alignToSlowest()`.

3.1.1 Local Time Scaling

With discretized time, the first and second derivatives are computed as

$$\begin{aligned}\dot{q}_i(t) &= \frac{q_i(t+\Delta t) - q_i(t)}{\Delta t}, \\ \ddot{q}_i(t) &= \frac{\dot{q}_i(t+\Delta t) - \dot{q}_i(t)}{\Delta t} \\ &= \frac{q_i(t+2\Delta t) - 2q_i(t+\Delta t) + q_i(t)}{\Delta t^2}.\end{aligned} \tag{3}$$

Given a maximum angular velocity $\dot{q}_{i,\max}$ and acceleration $\ddot{q}_{i,\max}$, if a velocity or acceleration limit is exceeded, we can enforce it by scaling the time locally. This is achieved by multiplying the time with a factor $\dot{q}_i(t)/\dot{q}_{i,\max}$ for velocity constraints and $\sqrt{\ddot{q}_i(t)/\ddot{q}_{i,\max}}$ for acceleration constraints. To avoid unnecessary scaling of the whole path, we confine this scaling locally by only considering regions where the constraint is violated. Given such a region with a duration of $t_>$, center t_c, a maximum value of $\dot{q}_i(t_{\max})$ or $\ddot{q}_i(t_{\max})$ respectively, the time scaling is given by a Gaussian with mean t_c and standard deviation $t_>$. This Gaussian is rescaled to have a minimum value of 1 and a maximum value of $\dot{q}_i(t_{\max})/\dot{q}_{i,\max}$ or $\sqrt{\ddot{q}_i(t_{\max})/\ddot{q}_{i,\max}}$ respectively. This reduces the acceleration and velocity locally to stay within the given bounds. The overall time scaling factor is then given by the maximum of all Gaussians for a given time step. Figure 5 shows an example of the time scaling for velocity and acceleration constraints.

```
$Q_{\text{all}}, T_{\text{all}} \leftarrow [\,], [\,]$;
for $i \in [1, n]$ do
    initialize $(T_{\min}, T_{\max})$;
    while $T_{\max} - T_{\min} > \Delta T$ do
        $T_{\text{mean}} \leftarrow (T_{\max} + T_{\min})/2$;
        initialize $(m_{\min}, m_{\max})$;
        while $m_{\max} - m_{\min} > 1$ do
            $m_{\text{mean}} \leftarrow \lfloor (m_{\max} + m_{\min})/2 \rfloor$;
            $Q \leftarrow \text{smRanFun}(T_{\text{mean}}, m_{\text{mean}})$;
            $\dot{Q}, \ddot{Q} \leftarrow \text{derivatives}(Q)$;
            if $2 * \text{std}(\dot{Q}) < \dot{q}_{i,\max}$ & $2 * \text{std}(\ddot{Q}) <$
              $\ddot{q}_{i,\max}$ then
                $T_{\min} \leftarrow T_{\text{mean}}$;
            else
                $T_{\max} \leftarrow T_{\text{mean}}$;
            end
        end
        $\hat{Q}, \hat{T} \leftarrow \text{localTimeScaling}(Q, T_{\text{mean}},$
          $\dot{q}_{i,\max}, \ddot{q}_{i,\max})$;
        $\tilde{Q}, \tilde{T} \leftarrow \text{quadraticClipping}(\hat{Q}, \hat{T}, q_{i,\min},$
          $q_{i,\max}, \ddot{q}_{i,\max})$;
        if $\tilde{T} < T$ then
            $T_{\min} \leftarrow T_{\text{mean}}$;
        else
            $T_{\max} \leftarrow T_{\text{mean}}$;
        end
    end
    $Q_{\text{all}}.\text{append}(\tilde{Q})$;
    $T_{\text{all}}.\text{append}(\tilde{T})$;
    $Q_{\text{all}}, T_{\text{all}} \leftarrow \text{alignToSlowest}(Q_{\text{all}}, T_{\text{all}})$;
    return $Q_{\text{all}}, T_{\text{all}}$;
end
```

Algorithm 1: Automatic sampling trajectory generation based on smooth random functions. The abbreviated method `smRanFun()` describes the generation of a smooth random function that is normalized according to the given angular constraints.

3.1.2 Quadratic Clipping

To enforce the position constraints, we replace the parts exceeding the angular constraint $q_{i,\max}$ by a quadratic function. The quadratic coefficient is given by the acceleration constraint $\ddot{q}_{i,\max}$. However, not only the exceeding points need to be replaced but also points with an angular velocity so high that it is guaranteed that the angular constraint will be exceeded due to the acceleration constraint. To determine these points, for each time step t we compute the necessary angular acceleration $\ddot{q}_{i,\text{stop}}(t) > 0$ to stop right at $q_{i,\max}$. We denote the corresponding time as $t_{\max}$. In the following considerations, $\hat{t} > t$ denotes a time step along the path of maximal deceleration. We compute the required acceleration value as

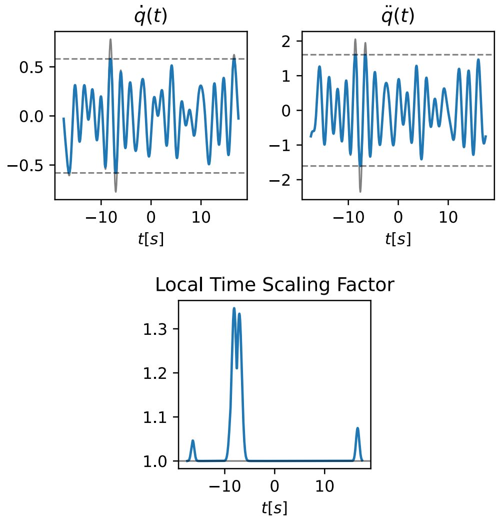


Figure 5: Local time scaling enforces the constraints on the first and second derivatives of the signal. For better visibility, the time axis is kept the same for all plots. The clipped gray line shows the original signal while the blue line shows the modified signal. The lower figure shows the ratio by which the time is locally delayed in order to constrain the derivatives.

$$q_i(\hat{t}) = q_i(t) + \dot{q}_i(t)\hat{t} - \frac{1}{2}\ddot{q}_{i,\text{stop}}(t)\hat{t}^2 \tag{4}$$

$$\dot{q}_i(t_{\max}) = \dot{q}_i(t) - \ddot{q}_{i,\text{stop}}(t)t_{\max} = 0$$

$$\implies t_{\max} = \frac{\dot{q}_i(t)}{\ddot{q}_{i,\text{stop}}(t)}$$

$$q_i(t_{max}) = q_{i,\max} = q_i(t) + \frac{\dot{q}_i(t)^2}{\ddot{q}_{i,\text{stop}}(t)} - \frac{\dot{q}_i(t)^2}{2\ddot{q}_{i,\text{stop}}(t)}$$

$$\implies \ddot{q}_{i,\text{stop}}(t) = \frac{\dot{q}_i(t)^2}{2(q_{i,\max} - q_i(t))}.$$

If $\ddot{q}_{i,\text{stop}}(t) > \ddot{q}_{i,\max}$ holds, we know that the position constraint will be exceeded even if maximum deceleration is used to reduce the velocity. Thus, for each of these regions, we find the last valid time step t_j before this region and the first valid time step t_k after this region. We then compute a quadratic function that starts at t_j with initial velocity $\dot{q}_i(t_j)$ and constant acceleration $\ddot{q}_{i,\text{stop}}(t_j)$ and ends at $t_{\max}$ with a velocity of 0. Symmetrically, we compute a second quadratic function that starts at t_k with initial velocity $\dot{q}_i(t_k)$ and constant acceleration $\ddot{q}_{i,\text{stop}}(t_k)$ and continues backward in time until it reaches $q_{i,\max}$ with a velocity of 0. Both functions meet at $q_{i,\max}$. For enforcing minimum values, this method is applied to the inverted input signal. Figure 6 demonstrates this method.

As shown in Figure 6, the early deceleration causes a gap between the two quadratic functions.

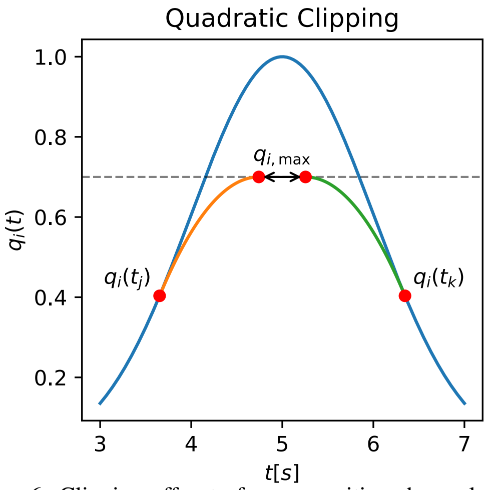


Figure 6: Clipping off out-of-range positions by replacing with quadratic polynomials.

Closing this gap causes a slight reduction in overall signal length. In combination with the artificial increase in signal length by the local time scaling, this necessitates the binary search for the optimal duration as shown in Algorithm 1.

In summary, the proposed algorithm finds a smooth random path for each joint that maximizes the overall randomness under the given constraints for the angles as well as angular velocity and acceleration. The distribution of a single generated trajectory is shown in Figure 7. It is directly visible how out-of-range values are shifted towards the limits of the allowed range. For the angle q_i, its distribution thus resembles a uniform distribution with peaks at its boundaries.

In Appendix A, we present an alternative approach to plan a feasible excitation trajectory by obtaining the Fourier coefficients of (2) directly from a nonlinear optimization problem.

3.2 Simulation

To test the feasibility of the approach, we created an experiment in the open-source software *Blender*[2]. The scene contains a single camera as well as a model of the *Kinova Gen2 7 DoF robot*[3] that has a spherical marker attached to the tip of the end-effector as a target, as seen in Figure 8a. The goal of our approach is to ensure the visibility of this target under arbitrary movements of the robot. Using the Blender Python

[2]https://www.blender.org/ (accessed January 29, 2024)

[3]https://github.com/Kinovarobotics/kinova-ros (accessed January 29, 2024)

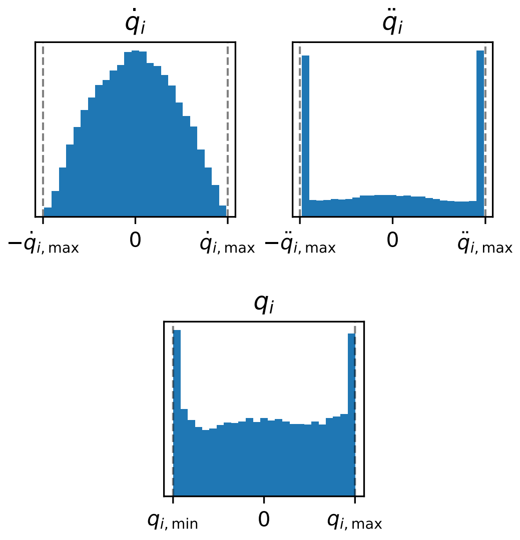
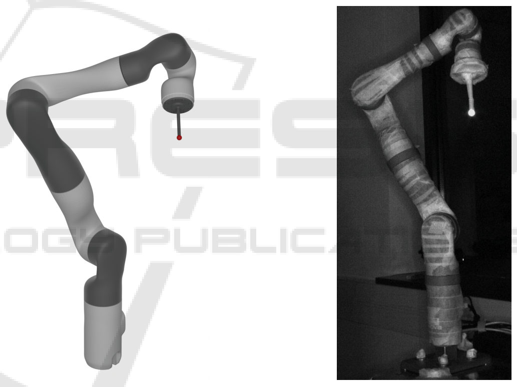


Figure 7: Distribution of a trajectory generated by the proposed path generation algorithm as well as the distribution of its first two derivatives.



(a) Simulation. (b) Laboratory Setup.

Figure 8: Simulated and real-world evaluation setups. The simulation is based on the publicly available robot model of the *Kinova*^{TM} Gen2 robotic arm.

API[4], we implemented a script that reads a trajectory and simulates the robot's pose for each time step. Then, we determined whether the marker was visible or hidden. This is done via a single traced ray, so it is not necessary to render the full scene, resulting in a simulation rate of $\approx 5 \cdot 10^3$ steps per second. The simplification is necessary for efficient sampling and results in a discontinuous binary output. Furthermore, it is possible to simulate any non-smooth trajectory, such as randomly sampled joint angles. This allows to create data for testing against the generated smooth trajectories.

[4]https://docs.blender.org/api/current/index.html (accessed January 29, 2024)

3.3 Real-World Setup

We built a setup very similar to the simulation in our laboratory, with the same Kinova robot, and attached a 3D printed version of the instrument with a small reflective sphere at its tip representing the point of interest as shown in Figure 8b. We then utilized a camera (Qualisys Arqus A9) with its own light source to automatically detect whether the marker is visible or not. This is achieved using thresholding visible marker size, resulting in a similar binary signal as in the simulation case. To prevent unwanted reflections on smooth surfaces of the robot, we covered it in not-reflective tape. We determined the limits for each of the robotic joints to prevent collisions during the automatic capturing process and generated a trajectory with a duration of 6 h. The trajectory is executed by the Kinova robot using its ROS interface. Therefore, a PD controller is designed to track the position and velocity reference of the trajectory sufficiently considering the actual joint position and velocity measurements. The resulting velocity commands are then commanded to the robot's joint drives.

3.4 Classification with Multilayer Perceptrons (MLPs)

There are several Machine Learning (ML) algorithms in the literature for supervised classification that have one or more disadvantages (Bishop and Nasrabadi, 2006; Cover and Hart, 1967; McNicholas, 2016; Cortes and Vapnik, 1995; Schölkopf and Smola, 2005). Problems can arise from the complexity of the model or the sensitivity of the hyperparameters. For example, Nearest Neighbor models (Cover and Hart, 1967) require all training points to be stored, which is not computationally feasible in our scenario. Other ML models, like Support Vector Machines (Cortes and Vapnik, 1995; Schölkopf and Smola, 2005)) or Gaussian Models for Classification (McNicholas, 2016) use only a subset of the data. Unfortunately, these models are highly dependent on hyperparameters and also require high computational resources when trained on raw data. In contrast to this, MLPs have shown the ability to represent highly complex functions (Hornik, 1991). With low layer counts the computational requirements for training are low while providing an efficient representation of the sampled space. For this reason, they are used in learning high-dimensional spaces, such as 5-dimensional scene representations for view synthesis in Neural Radiance Fields (NeRFs) (Mildenhall et al., 2021). Furthermore, due to their widespread availability in machine-learning frameworks, they are easily deployed with hardware acceleration, thus benefiting from a high degree of parallelization. Another crucial benefit is that it is computationally cheap to compute their gradient. This is necessary for integrating them as part of a larger optimization problem in the context of trajectory planning. One potential drawback of this method is that MLPs are struggling to learn high-frequency details. For the robot used in our scenario, this did not pose any problems due to the simplicity of its geometry. However, for mesh-like structures with fine detail, an approach based on a positional encoding would be beneficial (Mildenhall et al., 2021).

In our case, the chosen network architecture starts with an input layer with n inputs representing the robot's joint angles. A series of fully connected layers follows, each with a Rectified Linear Unit (ReLU) activation function that we chose for their computational efficiency. Finally, a fully connected layer combines the hidden layers' activations and passes them through a final single sigmoid activation function to map the final output to the interval $[0,1]$. For optimization, we chose the Adam optimizer (Kingma and Ba, 2014). All hyperparameters were determined via hyperparameter optimization using the Asynchronous Successive Halving Algorithm (ASHA) scheduler (Li et al., 2020) and the Optuna search algorithm (Akiba et al., 2019). As a loss function we used binary cross-entropy, which is a commonly used loss for binary classification problems

$$L(Y^{pred}, Y^{true}) = \frac{1}{n}\sum_{i=1}^{N} -\alpha_i \left[\, Y_i^{true} \log Y_i^{pred} + \left(1 - Y_i^{true}\right) \log\left(1 - Y_i^{pred}\right) \right]. \tag{5}$$

The balancing factor α_i is used to increase the contribution of underrepresented samples. With P being the number of positive samples and N being the number of negative samples, the factor is set to $\frac{P+N}{2P}$ for positive and to $\frac{P+N}{2N}$ for negative samples.

During the training, 20% of the data were used as validation data. The validation and training data are chosen as continuous parts of the trajectory, as random shuffling would with a high likelihood cause close training samples for every validation sample. To further reduce this risk, a buffer between the two segments with a length of 1% of the data was removed before training. Besides this evaluation data, for the simulated scenario, we created a test set of 10^6 uniformly sampled robot configurations to avoid overfitting on the validation data. All models were implemented in the PyTorch framework and trained on a consumer-level system with a Ryzen 3600 as a CPU, a GTX 1660 Ti as a GPU, and 16 GB of Memory. We

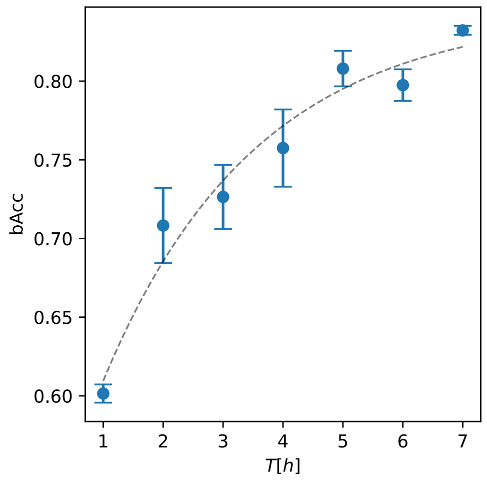


Figure 9: bAcc scores were measured on the test set for increasing trajectory durations. For better visualization, the dashed line shows a fit for an exponentially decaying function.

trained for $5 \cdot 10^4$ epochs but stopped early if no improvement was seen in the validation score for 10^3 epochs. In these conditions, training a single network takes approximately 1 min.

4 RESULTS

In this section, we show the training results for the MLPs with a focus on the required trajectory length, the hyperparameter tuning, and computational efficiency. Furthermore, we demonstrate with a short example how the MLP can be integrated into trajectory-planning to increase the overall visibility.

We initially established a baseline score by training on a simulated dataset of 10^6 uniformly sampled joint states. When trained until saturation, the network scored a balanced accuracy of 94% for predicting the correct visibility of the chosen target.

For smooth simulated trajectories, the results show increasing test scores as shown in Figure 9. After a steep increase at the beginning, the increase in gained balanced accuracy reduces at higher durations. Due to these diminishing returns, we chose six hours as the duration for all further experiments.

For a simulated trajectory with a duration of six hours, the hyperparameter optimization yielded the following results

- learning rate: $2 \cdot 10^{-3}$,
- number of hidden layers: 2,
- size of hidden layers: 78.

With these parameters, when trained until saturation on a simulated trajectory of 6 h, the model achieves a balanced accuracy of 85% on the uniformly sampled test set. A more detailed view of this result is shown by the confusion matrix in Table 1. Based on these results, the recall for the visible class

Table 1: Confusion matrix for the predictions of the MLP trained on a rendered 6 h smooth trajectory. The values are normalized by the total number of samples in the test set.

	Predicted Visible	Predicted Hidden
Visible	74.28%	12.34%
Hidden	2.07%	11.30%

is 85.75% and 84.53% for the hidden class. Furthermore, the overall ratio of visible samples in the dataset is 86.63%. The measured precision is 97.29% for the visible and 47.80% for the hidden class.

To confirm these simulation results, we captured the same 6 h trajectory on our real-world setup (see Figure 8b) and trained the MLP on this data until no further improvement was noticed. We obtained slightly higher bAcc scores of 90% in this case.

When evaluating the model's latency on $1 \cdot 10^3$ samples, we found the results shown in Table 2. Furthermore, every measured latency stayed below 1 ms.

Table 2: Inference and gradient computation times for the proposed MLP.

	Forward-Pass	**Backward-Pass**
mean	78 µs	391 µs
5% low	95 µs	478 µs
1% low	150 µs	626 µs

To show the feasibility of the proposed approach, we chose a simple test case: Increasing the visibility under the constraint of a stable end-effector. We modeled this via the forward kinematics function $e(\mathbf{Q}) : \mathbb{R}^n \to \mathbb{R}^3$, that provides the position of the end-effector for a given combination of joint angles $\mathbf{Q}$. Furthermore, the trained MLP is denoted as the function $v(\mathbf{Q})$ that predicts the visibility of the robot at a given state. For a starting configuration $\mathbf{Q}_{\text{start}}$ we find a trajectory that maximizes the visibility under the constraint of a stable end-effector by reducing the following loss

$$L(\mathbf{Q}) = \gamma_e ||e(\mathbf{Q}) - e(\mathbf{Q}_{\text{start}})|| + \gamma_v (1 - v(\mathbf{Q})). \quad (6)$$

The first part of this loss reduces deviations in the end-effector position from the initial configuration while the second part increases the overall visibility using the trained MLP. The weight factors γ_e and γ_v allow us to put more emphasis on one of the two aspects during optimization. In our case, we chose $\gamma_e = 10^4$ and $\gamma_v = 1$ to ensure a stable end-effector position.

The trajectory is then obtained by repeated computations of the gradient $\nabla_{\mathbf{Q}}$ and changing $\mathbf{Q}$ in the

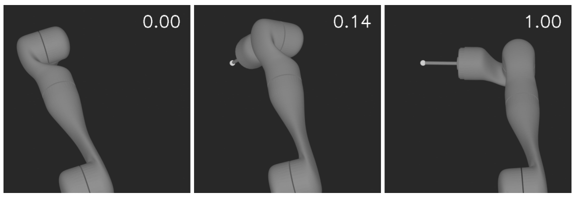


Figure 10: Renderings of a trajectory that maximizes the visibility of the target at the end-effector (bright sphere) while keeping it stable. The corresponding predicted visibility scores are shown in the upper right.

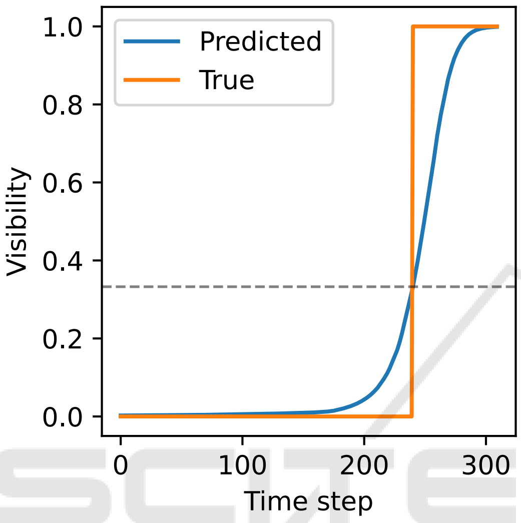


Figure 11: Predicted and rendered visibility along the generated trajectory.

opposite direction of this gradient. The generated trajectory is shown qualitatively in Figure 10. As shown in Figure 11, the predicted visibility increases smoothly and corresponds to the measured visibility. The object becomes visible at a predicted visibility value of 0.33.

5 DISCUSSION

As shown by the 94% bAcc score on the synthetic uniformly sampled dataset, MLPs are a valid choice for learning the complex occlusion behavior of robotic systems. This score serves as a benchmark for smooth random functions. As Figure 9 shows, the added variability in the data from an increased trajectory duration benefits the training results. However, even considering MLPs trained on 6 h trajectories with fine-tuned hyperparameters, there is still a difference of 9 percentage points in the bAcc score. This shows that even though the smooth random functions slowly approach the benchmark score, this method of sampling does not fully reach the desired score at durations that are feasible to run in real-world applications. When investigating the results further, we found that the recall predicting either the visible or the hidden class was close to the measured bAcc of 85%, however, their respective recall is much higher for the visible class. This can largely be attributed to the high class imbalance in our dataset, which highly favors the visible class with 86.63%. The re-balancing factors caused by this imbalance drastically increase the contribution of the hidden class to the overall loss, which leads to the relatively high number of samples falsely classified as hidden.

One major limitation of our sampling method is that there is no feedback from the measured visibility to the sampling routine. This has the advantage that it allows for both components to run on separate hardware without the need for a real-time interconnection. While the separation of surveillance and control systems might represent reality in many scenarios, there are possibly important performance gains to be found without this limitation. Furthermore, such an approach would remove the need for synchronization between the two systems.

Another limitation of the proposed method is its reliance on a stationary camera. The MLP's approximated function depends highly on the camera's pose, moving the camera requires retraining the model. At the same time, the model generally does not depend on the camera's intrinsic. For this reason, changes in distortion or focal length do not require resampling and retraining the model.

An interesting finding is that the model's performance seems to be slightly higher on the real-world data than on the rendered training data. This could possibly be attributed to the small differences between the real and the simulated scene. One of the parameters that could have influenced this difference is the positioning of the camera, which differs between simulation and reality (see Figure 8).

Finally, we could show that the proposed method of using MLPs is computationally efficient enough for the application in real-time trajectory planning algorithms. Inference and gradient computation in the sub-ms latency range allows for real-time prediction and optimization. As we have shown in the example application, the trained MLP is easily integrated into trajectory planning and allows for the continuous optimization of the robot's observability under chosen constraints.

6 CONCLUSION AND OUTLOOK

In this paper, we proposed a method for automatically improving the visual observability of robotic systems. This method relies heavily on our two main contributions, the automatic sampling of the training space via modified smooth random functions as well as the training and evaluation of lightweight MLPs to represent the occlusion function. While there is still room for improvement due to the measured performance gap between independently sampled data and smooth trajectories, we could show that this approach works well in the simulation and real-world experiments and consistently shows high validation scores. We further showed the modular integration in a trajectory-planning example, where it successfully increased the visibility under given constraints.

There are several directions for further research, the most promising being a smart coupling of the trajectory generation with live feedback from the camera system. This would potentially allow for more efficient approaches to sampling the boundary between visible and hidden states, thus reducing the dimensionality of the problem. Another promising approach would be to disentangle camera pose, robot geometry, and robot kinematics into several learned components of a combined system. This would possibly allow it to be more resilient against minor changes in the scene and would only require minor retraining upon repositioning the camera. Another research direction is investigating different robot geometries and material properties. This could include semi-translucent and reflective surfaces as well as the addition of positional encodings for handling more complex joint geometries.

ACKNOWLEDGEMENT

This research is funded through the Project ”OP der Zukunft” within the funding program by Europäischen Fonds für Regionale Entwicklung (REACT-EU).

REFERENCES

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Ayvaci, A., Raptis, M., and Soatto, S. (2012). Sparse occlusion detection with optical flow. *International journal of computer vision*, 97:322–338.

Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*, volume 4. Springer.

Boroushaki, T., Leng, J., Clester, I., Rodriguez, A., and Adib, F. (2021). Robotic grasping of fully-occluded objects using rf perception. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 923–929. IEEE.

Brodersen, K. H., Ong, C. S., Stephan, K. E., and Buhmann, J. M. (2010). The balanced accuracy and its posterior distribution. In *2010 20th international conference on pattern recognition*, pages 3121–3124. IEEE.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20:273–297.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

Danielczuk, M., Angelova, A., Vanhoucke, V., and Goldberg, K. (2020). X-ray: Mechanical search for an occluded object by minimizing support of learned occupancy distributions. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9577–9584. IEEE.

Fan, X., Simmons-Edler, R., Lee, D., Jackel, L., Howard, R., and Lee, D. (2021). Aurasense: Robot collision avoidance by full surface proximity detection. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1763–1770. IEEE.

Filip, S., Javeed, A., and Trefethen, L. N. (2019). Smooth random functions, random odes, and gaussian processes. *SIAM Review*, 61(1):185–205.

Gandhi, D. and Cervera, E. (2003). Sensor covering of a robot arm for collision avoidance. In *SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483)*, volume 5, pages 4951–4955. IEEE.

Guizzo, E. (2019). By leaps and bounds: An exclusive look at how boston dynamics is redefining robot agility. *IEEE Spectrum*, 56(12):34–39.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-Tzur, J., Hardt, M., Recht, B., and Talwalkar, A. (2020). A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246.

Liu, Y., Li, Y., Zhuang, Z., and Song, T. (2020). Improvement of robot accuracy with an optical tracking system. *Sensors*, 20(21):6341.

McNicholas, P. D. (2016). *Mixture model-based classification*. CRC press.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2021). Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106.

Park, K.-J. (2006). Fourier-based optimal excitation trajectories for the dynamic identification of robots. *Robotica*, 24(5):625–633.

Rijsbergen, C. v. (1979). Information retrieval 2nd ed buttersworth. *London [Google Scholar]*, 115.

Schölkopf, B. and Smola, A. (2005). Support vector machines and kernel algorithms. In *Encyclopedia of Biostatistics*, pages 5328–5335. Wiley.

Shintemirov, A., Taunyazov, T., Omarali, B., Nurbayeva, A., Kim, A., Bukeyev, A., and Rubagotti, M. (2020). An open-source 7-dof wireless human arm motion-tracking system for use in robotics research. *Sensors*, 20(11):3082.

Silva, C. and Santos-Victor, J. (2001). Motion from occlusions. *Robotics and Autonomous Systems*, 35(3-4):153–162.

Stürz, Y. R., Affolter, L. M., and Smith, R. S. (2017). Parameter identification of the kuka lbr iiwa robot including constraints on physical feasibility. *IFAC-PapersOnLine*, 50(1):6863–6868. 20th IFAC World Congress.

Swevers, J., Ganseman, C., De Schutter, J., and Van Brussel, H. (1997a). Generation of Periodic Trajectories for Optimal Robot Excitation. *Journal of Manufacturing Science and Engineering*, 119(4A):611–615.

Swevers, J., Ganseman, C., Tukel, D., de Schutter, J., and Van Brussel, H. (1997b). Optimal robot excitation and identification. *IEEE Transactions on Robotics and Automation*, 13(5):730–740.

Vedaei, S. S. and Wahid, K. A. (2021). A localization method for wireless capsule endoscopy using side wall cameras and imu sensor. *Scientific reports*, 11(1):11204.

Zheng, Z., Ma, Y., Zheng, H., Gu, Y., and Lin, M. (2018). Industrial part localization and grasping using a robotic arm guided by 2d monocular vision. *Industrial Robot: An International Journal*.

APPENDIX

A OPTIMIZATION-BASED TRAJECTORY GENERATION

An alternative approach to the trajectory generation described in Section 3.1 is to obtain the Fourier coefficients for (2) by solving an optimization problem to ensure that the trajectory provides a sufficient excitation of the robot's joint space. This removes the required post-processing step while ensuring that the robot's kinematic and dynamic limits are satisfied. Therefore, we define a cost function that maximizes the energy of the trajectory for each joint, while minimizing the cross-correlation with the trajectories of the other joints. According to Parseval's theorem, the energy of a signal is identical to the sum of its squared Fourier coefficients. These considerations lead to the cost function

$$J(\pi) = \sum_{i=1}^{n} \left(-w_{\mathrm{e}} \sum_{k=0}^{m} \left(a_{i,k}^2 + b_{i,k}^2\right) + w_{\mathrm{c}} \sum_{\substack{j=1 \\ j \neq i}}^{n} \rho(q_j(t), q_i(t)) \right) \tag{7}$$

where $\rho(q_j(t), q_i(t))$ denotes the Pearson correlation coefficient between the two Fourier series (2). The influences of the energy and cross-correlation terms are weighted with the positive factors w_{e} and w_{c}. The vector π contains the Fourier coefficients as optimization variables. Including the above-described kinematic and dynamic constraints gives the nonlinear optimization problem

$$\pi = \arg\min_{\pi} J(\pi) \tag{8a}$$

s.t

$$q_{i,\min} \leq a_{i,0} + \sum_{k=1}^{m} \sqrt{a_{i,k}^2 + b_{i,k}^2} \leq q_{i,\max} \tag{8b}$$

$$\sum_{k=1}^{m} \frac{2\pi k}{T} \sqrt{a_{i,k}^2 + b_{i,k}^2} \leq \dot{q}_{i,\max} \tag{8c}$$

$$\sum_{k=1}^{m} \left(\frac{2\pi k}{T}\right)^2 \sqrt{a_{i,k}^2 + b_{i,k}^2} \leq \ddot{q}_{i,\max} \tag{8d}$$

$$b_{i,0} = 0 \tag{8e}$$

where the constraints (8b)–(8e) must be satisfied for all joints $i \in \{1, \dots, n\}$.

The nonlinear optimization problem (8) is implemented in *Python* with *NumPy*. It is solved by the `minimize` function from *SciPy* using the *constrained optimization by linear approximation (COBYLA)* algorithm. The Jacobians of the cost function and the nonlinear constraints are estimated using numerical differentiation.

Kinematic constraints in the Cartesian workspace of the robot can be easily included in the optimization problem by using the forward kinematic of the robot. The presented approach works with up to 50 modes, while the coverage of the state space decreases with a higher number of modes. This may be due to the approximations (8b)–(8d), which overestimate the maximum amplitudes for position, velocity, and acceleration of the trajectories. This can be resolved by an equidistant sampling of the trajectories and defining the corresponding linear inequality constraints. The drawback of this method is that the computational cost depends largely on the accuracy of the sampling, while the constraints may be violated between two sampling points.