UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR ELECTRICAL
ENGINEERING IN MEDICINE

Camera-Supported Uncertainty Awareness for Guided Real-World Reinforcement Learning

Kameraunterstütztes Unsicherheitsbewusstsein für geführtes Reinforcement Learning in der realen Welt

Masterarbeit

verfasst am
Institut für Medizinische Elektrotechnik

im Rahmen des Studiengangs
Robotik und Autonome Systeme
der Universität zu Lübeck

vorgelegt von
Lugh Martensen

ausgegeben und betreut von
Prof. Dr. Georg Schildbach

mit Unterstützung von
Fabian Domberg

Christoph Willibald,
Abhishek Padalkar

Lübeck, den 30. April 2026

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Für Korrekturen und Verschönerungen der Satzstrukturen wurden in dieser Arbeit die KI-basierten Tools DeepL Write und ChatGPT (OpenAI) benutzt.

_____

Lugh Martensen

Zusammenfassung

Learning from Demonstration (LfD) hat sich als intuitive und benutzerfreundliche Methode zur Programmierung von Robotern etabliert, die es ermöglicht, komplexes Verhalten ohne detailliertes Umgebungsmodell zu imitieren und anzupassen. In Szenarien, in denen Demonstrationen die Dynamik von Aufgaben nicht vollständig erfassen, insbesondere bei kontaktreichen industriellen Aufgaben, kann LfD jedoch unzuverlässig sein, da es schlecht auf unbekannte Zustände generalisiert.

Reinforcement Learning (RL) hat sich als vielversprechende Lösung herausgestellt, indem es eine LfD-Referenztrajektorie zustandsabhängig korrigiert. So entsteht ein Verhalten, das Demonstrationen folgt und sich gleichzeitig dynamisch anpasst, um die Zuverlässigkeit des Systems zu erhöhen. In den letzten Jahren wurde ein neuer Ansatz namens Kernelized Guided Reinforcement Learning (KGRL) vorgestellt, der diese Idee erweitert, indem der Explorationsraum des RL-Agenten auf Grundlage der probabilistischen Eigenschaften des demonstrierten Verhaltens geformt wird. Indem der Roboter darauf beschränkt wird, nur Zustände zu explorieren, die nahe an zuvor beobachteten Zuständen liegen, werden Sicherheit und Sample-Effizient beim Einsatz von RL in realen Anwendungen adressiert. Eine Voraussetzung für KGRL ist jedoch, dass die in den Demonstrationen kodierte Unsicherheit die Variabilität der Aufgabe ausreichend erfasst, was in der Realität oft nicht der Fall ist.

In dieser Arbeit wird eine Erweiterung von KGRL vorgestellt, die als External-Uncertainty KGRL (EU-KGRL) bezeichnet wird. Der Kerngedanke besteht darin, externe Unsicherheiten, etwa aus einem Kamerasystem, in Form von Kovarianzen mit einzubeziehen. Dafür wurde eine markerlose, kamerabasierte Pipeline zur Posenschätzung implementiert, die eine Genauigkeit im Millimeterbereich bei der Positionsschätzung erreicht. Ihre Genauigkeit und Präzision werden analysiert, um die daraus resultierende Unsicherheit zu charakterisieren.

Die vorgeschlagene Erweiterung wurde sowohl in Simulation als auch auf einem realen Roboter für Szenarien mit unsicherer Zielpose evaluiert. In Simulation zeigt EU-KGRL die Fähigkeit, Aufgaben durch ein strukturiertes Explorieren in relevante Richtungen zu lösen, die von KGRL allein nicht bewältigt werden können. In Experimenten in der echten Welt wurde die Methode für das Einstecken eines Ethernet-Steckers untersucht, wobei die Position des Zielgeräts mithilfe des implementierten Posenschätzers bestimmt wurde. Die Ergebnisse zeigen, dass EU-KGRL in bestimmten Szenarien Positionsungenauigkeiten effektiver kompensieren kann als KGRL. Gleichzeitig wurden die Herausforderungen beim Einsatz von EU-KGRL in realen Anwendungen deutlich, da der Ansatz auf ein genaues Unsicherheitsmodell angewiesen ist.

Abstract

Learning from Demonstration (LfD) has become an intuitive and user-friendly method for robot programming. It enables robots to imitate complex behaviors and adapt them to task-specific constraints without requiring detailed models of the environment. However, in scenarios where demonstrations do not fully capture all task dynamics, LfD can become unreliable. For contact-rich tasks, which are common in manufacturing, demonstration-based policies often struggle to generalize due to out-of-distribution states.

Reinforcement Learning (RL) has emerged as a promising solution. By training an RL policy to correct a baseline LfD trajectory based on the robot's state, the resulting behavior adheres to the demonstrations while dynamically adapting to its environment to improve reliability. In recent work, a new approach called Kernelized Guided Reinforcement Learning (KGRL) extends this idea by shaping the RL agent's exploration space based on the probabilistic properties of the demonstrated behavior. Restricting the robot to only explore states that are close to previously observed ones addresses common concerns about safety and sample efficiency when applying RL to real-world settings. However, KGRL assumes that the uncertainty encoded in demonstrations sufficiently captures the variability of the task, which is often not the case in practical scenarios.

In this thesis, an extension of KGRL, termed External-Uncertainty KGRL (EU-KGRL), is proposed. The key idea is to incorporate externally-provided covariance into the framework, allowing the exploration space to be shaped by uncertainty sources, such as perception systems, in addition to demonstrations. To this end, a markerless, camera-based pose estimation pipeline is implemented, which achieves millimeter-level accuracy in position estimation. Its accuracy and precision are analyzed to characterize the resulting uncertainty.

The proposed extension is evaluated both in simulation and on a real robotic system, for scenarios with target pose uncertainty. In simulation, EU-KGRL demonstrates the ability to solve tasks that KGRL alone cannot address, by enabling structured exploration in task-relevant directions. In real-world experiments, the method is applied to an Ethernet connector-insertion task, in which the position of the target device is estimated using the implemented pose estimator. The results show that, in certain settings, EU-KGRL can compensate for positional inaccuracies more effectively than KGRL. At the same time, the challenges of deploying EU-KGRL in a real-world setting are highlighted, as it relies on an accurate uncertainty model.

Acknowledgements

I would like to express my sincere gratitude to all those who have supported me throughout the course of my master's thesis.

First and foremost, I would like to thank Christoph Willibald for his supervision and support, not only for this thesis but also for his guidance regarding my professional development beyond this work. I am also grateful to Abhishek Padalkar for his supervision and valuable input. My thanks also go to Fabian Domberg for his supervision and practical advice.

I am also thankful to Thomas Eiband, Korbinian Nottensteiner, and João Silvério for their assistance and support during the final months of this project, as well as Anne Reichert for her valuable assistance and support in the area of computer vision.

I would like to extend my appreciation to Neal Y. Lii for his support in matters beyond the technical scope of this work and for always being approachable and willing to listen.

Furthermore, I would like to thank everyone who took the time to give me feedback on my work, in particular Lisa Schönherr, Tim Holthuijsen, Leonardo Bonanno, and Riya Thomas.

A very special and heartfelt thank you goes to Riya Thomas, whose support extended far beyond this thesis and who provided me with invaluable encouragement throughout this time. Finally, I am deeply grateful to my parents for their unwavering emotional support.

Contents

1

Introduction

Despite economic incentives, fine manipulation tasks like assembly have not been automated on a large scale in manufacturing [1] [2]. Tight tolerances and component variability in high-precision assembly scenarios require specialized fixtures, end-effectors, and compliance mechanisms, which can become uneconomical for small batch sizes or complex use cases. At the same time, modern robotic arms can measure contact forces, enabling precise and safe execution of in-contact tasks and allowing for safe human-robot interactions [3] [4].

To harness the potential of robots for automating fine-mechanical assembly tasks, dynamic control algorithms are needed to account for workspace variations and uncertainties. However, tuning such algorithms for different use cases is non-trivial. While minimizing contact forces to reduce wear and tear on components might be desirable, interaction with the environment remains necessary to perform manufacturing tasks. Describing in-contact tasks parametrically is challenging, as humans typically perform them intuitively.

1.1 Motivation

Learning from Demonstration (LfD) is an intuitive and user-friendly method to generate robot motions [5]. An expert gives one or multiple demonstrations, for example, by kinesthetic teaching. The LfD algorithm then enables the robot to imitate and adapt the demonstrated motions, possibly considering contact forces and other sensor data that was recorded during demonstration. Frameworks like Dynamic Movement Primitives (DMPs) [6] and Kernelized Movement Primitives (KMPs) [7] also allow to generalize to new situations and to avoid obstacles. However, for in-contact tasks, in which the recorded demonstrations often do not fully describe the complex task dynamics, these frameworks struggle due to model imperfections and when receiving out-of-distribution input states that were not present during demonstration [8] [5] .

Kernelized Guided Reinforcement Learning (KGRL) [8] solves this problem by extending the LfD paradigm with a reactive policy trained to consider the state of the robot and its environment. The algorithm encodes a set of user demonstrations, which were recorded through kinesthetic teaching, as probabilistic movement primitives using

KMPs [7]. Based on those, a mean baseline trajectory is generated that imitates the trajectories obtained through demonstration. A Reinforcement Learning (RL) policy then learns to correct this trajectory based on the state of the robot, considering interactions with the environment through contact force sensors. Exploration for the RL policy is restricted using linear safety constraints [9] and shaped by the variance that was observed during the demonstrations. Thus, KGRL guides exploration to regions of the state-action space that are considered safe and relevant, improving both sample efficiency and robustness. This enables reinforcement learning directly on a real robotic system, eliminating the need for simulation-based pretraining and thus mitigating the sim-to-real gap. KGRL has been successfully applied to the task of inserting a BNC connector [1], demonstrating its capability to work reliably in high-precision, contact-rich assembly scenarios.

In the scenario of connector-insertion examined in [8], variations are primarily caused by calibration inaccuracies, robot kinematics, and impedance controller offsets, as well as minor, unpredictable movements when grasping the connector. The workspace configuration is static and known in advance, including the precise locations of all objects the robot needs to interact with. However, this assumption may not be valid in many real-world scenarios. The objects the robot needs to interact with might have to be localized first. In dynamic environments, unpredictable reconfigurations of the workspace are possible. In the context of manufacturing, this challenge could arise in an interactive setting in which the robot has to collaborate with a human worker, or when the robot is faced with an anomaly it must recover from.

Despite the advantages of KGRL, it relies on a key assumption: that the uncertainty present in the demonstrations adequately captures the uncertainty of the task and environment. In real-world applications, this assumption is often violated. For instance, when perception modules such as vision-based pose estimators are used, significant uncertainty may arise from sensor noise. This external uncertainty is not reflected in the demonstration data and therefore cannot be accounted for by standard KGRL. As a result, KGRL may exhibit suboptimal behavior in scenarios where external uncertainty dominates. In these cases, the algorithm may either overconfidently exploit inaccurate estimates or fail to adapt its exploration strategy to the true uncertainty of the environment. Guiding exploration based only on demonstrations limits the approach by design and makes it inherently inflexible.

1.2 Contributions

To address this limitation, this thesis proposes an extension to KGRL that explicitly incorporates externally-provided uncertainty into the learning process. The proposed method, termed External-Uncertainty Kernelized Guided Reinforcement Learning (EU-KGRL), replaces the demonstration-derived uncertainty with covariance information obtained from external sources. In scenarios in which information about such an external covariance is available, the extended algorithm can better reflect the actual uncertainty present in the task and adapt its behavior accordingly.

Both simulation and real-world experiments are conducted to assess the performance of EU-KGRL in comparison to the original KGRL formulation. Similar to the KGRL experiments in [8], the proposed approach is evaluated using a goal-reaching task in a simulated environment and a robotic connector-insertion task in the real world. In both cases, only an estimate of the target pose is given, together with the associated uncertainty. In the simulation, the target pose is a point the robot has to reach, and is sampled from a distribution. In the real-world setup, it is the pose of a connector port, and is provided by a camera-based pose estimator. The goal is to make EU-KGRL applicable in situations where the workspace configuration is not known precisely. In such cases, the target pose has to be determined first.

While modern camera-based pose estimation algorithms achieve high precision, achieving sub-millimeter accuracy remains a challenge in many scenarios [10], which can conflict with the tight tolerances in manufacturing [2]. This problem could be solved by adopting the idea of KGRL to train a reactive RL policy that leverages contact force sensors. Similar to how a human worker can intuitively perform fine-mechanical tasks, a robot could use tactile feedback to adjust its movements without knowing the exact target pose. The original KGRL formulation assumes that the variations present during deployment remain within the variability observed in demonstrations. When the external variance from pose estimation exceeds the variance present in demonstrations, the KGRL framework does not readily adapt. This can lead to the complete failure of the learned policy when it is applied to unseen target poses without retraining. Replacing the true target pose with a possibly imperfect estimate is a realistic real-world scenario in which KGRL might fail, and is therefore used to investigate the potential of EU-KGRL.

Instead of the BNC-insertion task solved with KGRL in [8], the task of inserting an Ethernet connector was considered to evaluate the proposed approach. Compared to BNC connectors, Ethernet connectors typically exhibit larger mechanical tolerances, which simplifies the insertion process and makes it a suitable testbed for studying uncertainty-aware learning methods. Furthermore, Ethernet-insertion is also a task in the NIST[1] benchmark [1] that motivated the evaluation of KGRL in a setting with a BNC connector [8].

For the purpose of this work, the problem is reduced to estimating the position of the target, rather than its full six-degree-of-freedom (6D) pose. This simplification is justified by the structure of the test setup, in that the orientation of the connector is constrained by the environment and remains approximately constant during insertion. Consequently, the dominant source of uncertainty lies in the positional estimate, making it a meaningful and sufficient focus for evaluating the proposed method. By reducing the problem dimensionality, the influence of externally-provided covariance on the policy can also be analyzed more directly, without introducing additional complexity from orientation estimation.

[1] National Institute of Standards and Technology [1].

The main contributions of this thesis are as follows:

Extending KGRL:
: The formulation of External-Uncertainty Kernelized Guided Reinforcement Learning (EU-KGRL), extending KGRL by incorporating externally-provided covariance into the learning process

Pose Estimator Implementation:
: The implementation of a markerless, vision-based pose estimation pipeline for robotic manipulation tasks

Pose Estimator Analysis:
: An analysis of the uncertainty characteristics of the pose estimation system, with a focus on the resulting covariance in target position estimates

Pose Estimator Integration:
: The integration of pose estimation uncertainty into the EU-KGRL framework, enabling the algorithm to account for perception-induced uncertainty during learning

Evaluation in Simulation:
: A proof-of-concept evaluation of EU-KGRL in a modified version of the simulation used in [8]

Evaluation on Robot:
: An empirical evaluation of EU-KGRL on a real-world robotic connector-insertion task

Comparison:
: A comparative analysis of EU-KGRL and KGRL in both simulation and real-world settings

The proposed extension to KGRL aims to enhance the flexibility and robustness of guided RL, mitigating the need for restrictive assumptions about demonstration variance. This enables its application in more realistic settings where perception is imperfect. By explicitly modeling and incorporating external variance from pose estimation into the learning process, this thesis positions guided RL as a more practical tool for real-world robotic manipulation, capable of adapting to variations that are larger than those observed during demonstrations.

The remainder of this thesis is structured as follows. Chapter 2 reviews related work in guided reinforcement learning for robotic manipulation and pose estimation. Chapter 3 introduces the necessary background on movement primitives and KGRL. Chapter 4 describes the hardware and software used for this work and how the system was set up. Chapter 5 presents the proposed methodology, including the construction of the pose estimation pipeline, the extension of the simulation, and the EU-KGRL formulation. Chapter 6 examines the accuracy of the pose estimator, and evaluates KGRL and EU-KGRL in simulation and real-world experiments. Finally, Chapters 7 and 8 discuss the results and conclude the thesis.

2

Related Work

This chapter presents related work on guided reinforcement learning, which enables robots to be trained in the real world. In this thesis, such an approach was tested on a task with target position uncertainty, induced by a markerless camera-based pose estimator. Therefore, relevant works in the domain of pose estimation are presented as well.

2.1 Guided Real-World Reinforcement Learning

Applying reinforcement learning (RL) to real-world robotic systems remains a challenging problem. RL algorithms are typically developed in simulation, due to a large number of required training episodes, and potentially dangerous or unexpected robot behavior during exploration [11]. However, despite advances in realism, the gap between simulation and reality still remains a problem when transferring policies that were learned in software to physical systems, especially for contact-rich tasks [11] [12].

Therefore, similar to KGRL [8], recent work explores the idea of modifying the exploration space of the robot to increase safety and sample-efficiency. In [12], the authors examine the use of Shared Control Templates (SCT) for RL. SCTs use transition functions to switch between states, which represent different phases of a task. Each state has predefined input mappings to convert low-dimensional control inputs to robot end-effector motion. Active constraints ensure that the generated output for a state stays within a safe region. The template ensures safe and stable execution, while the RL agent is allowed to modulate the behavior within defined limits. SCTs are particularly effective for in-contact manipulation tasks, and were used to successfully learn peg-in-hole component insertion directly on a real robot in under 70 episodes (corresponding to around 17 minutes) [12]. While no detailed model of the system dynamics is necessary, input mappings and transition functions have to be specified in advance, requiring expert knowledge about the task.

In [13], the authors propose a framework that combines Dynamic Movement Primitives (DMPs) [6], which is a widely used LfD method [14] [15] [7], with reinforcement learning by introducing a residual correction policy. The DMP encodes a baseline behavior extracted from human demonstrations, while an RL agent predicts task-space corrections to adapt to environmental uncertainties (e.g., friction, misalignment) without

requiring explicit contact models. In contrast to learning a full control policy, the residual formulation restricts learning to deviations from the demonstrated motion, significantly reducing the complexity of the problem and improving sample efficiency.

The method from [13] is evaluated both in simulation and on a real robotic system using a 7-DoF Franka Panda arm [13]. In the real-world experiments, the authors consider several insertion tasks, including Ethernet connector-insertion, which is particularly relevant for this thesis. For this task, start positions were sampled with translation offsets of up to ± 1 cm from the demonstrated pose in every episode, and orientations of up to $\pm$ 40 °. After training for 500 episodes on the real-world system (corresponding to approximately two hours), the residual approach outperforms the original DMP policy, achieving a success rate of around 70%. However, secondary objectives, such as minimizing contact forces with the environment, are not taken into account in this particular experiment.

These works highlight the effectiveness of combining reinforcement learning with task-specific information derived from demonstrations or predefined control policies. By constraining the learning problem to a meaningful subspace, these methods improve safety and sample efficiency, enabling real-world deployment. However, they do not incorporate uncertainty to shape the exploration process. This limitation motivates the approach taken in this thesis, where uncertainty, both from demonstrations and external sources, is used to guide learning.

2.2 Camera Pose Estimation

Depending on the use case and available hardware, a variety of different pose estimation strategies can be applied. In [2], relevant approaches for electrical connector socket pose estimation are divided into three categories:

Voting-based Every pixel or 3D point in a camera's point of view is either matched to an estimated 6D object coordinate or to a correspondence on a known model. A voting scheme then determines the final object pose estimate. This approach is especially effective when 3D point clouds are available. However, stereo vision systems that are necessary to generate such point clouds do not perform well on metallic and texture-poor objects.

Template-based Using a reference template of the object, a pose estimate is found by choosing the pose that results in the best match in a target image. This approach is argued to require large computing power and to perform poorly on symmetric, rounded, and textureless metallic objects.

Correspondence-based Multiple correspondences between an object in the scene and its 3D model are determined, which can be used to calculate the object's 6D pose with a Perspective-n-Point (PnP) algorithm [16]. A simple method to find correspondences is to use markers, but the accuracy depends on how precisely these markers were mounted. Other methods use naturally occurring features in the image, but their performance varies based on the texture-richness and symmetry of the object. Neural network-based approaches can be used to improve accuracy by training a network to find feature points in an image, or to locate labeled keypoints.

To achieve reliable and precise pose estimation for connector sockets, which are often shiny and textureless, the authors of [2] propose a network-based approach for monochrome 2D images. Their architecture incorporates a modified version of U-Net [17] that generates heatmaps. These heatmaps are transformed into keypoint coordinates using DSNT (Differentiable Spatial to Numerical Transform) [18], a fully differentiable layer that adds no additional trainable parameters. Using the keypoint coordinates together with corresponding 3D coordinates extracted from a 3D model of the target object, a PnP algorithm calculates the socket pose. The network is trained on eight images of the connector port opening, which are captured in the real world with a camera mounted onto a robotic arm. Each image is manually labeled with four keypoints. The positional pose estimation accuracy was tested on six standardized electrical connector ports with varying material, shape, size, and mating tolerance (BNC, C14, N-type, SMA, USB-A, and USB-C).

The architecture proposed in [2] managed to achieve sub-millimeter accuracy for most connector types and outperforms an industrial template-based state-of-the-art algorithm. Additionally, the practical usability of the network was tested in a robotic mating experiment, in which a high success rate was achieved for the BNC connector type. While these results are promising, it should be noted that this algorithm is limited to the estimation of position in horizontal and vertical directions, reducing the problem to two dimensions. The authors argue that most sockets do not show visible variation along the image plane's normal direction, and variation in depth estimation could be compensated by the robot's force controller. Another limitation is that this position estimator can only be used for refinement. In the training images, the port pose randomly deviates up to 2° and 8 millimeters from the initial pose, making the trained network unsuitable for finding objects in a larger scene. The need for manual training data acquisition and labeling also makes this approach less flexible, as human assistance is necessary to train the network for a new connector type.

A different deep-learning approach that solves these problems is presented in [10]. The so-called *Augmented Autoencoder* (AAE) is trained entirely on synthetic data that is obtained by rendering the object's 3D model in different orientations with varying lighting conditions and backgrounds. During training, the encoder-decoder architecture learns to reconstruct the rendered images. As this is done for a vast number of different orientations, the network learns to encode the information about the object orientation in its latent representation. After training, a latent codebook for different orientations is created, which maps the latent representations to object orientations. At test time, an object detector, that can be trained on rendered images as well, localizes the target object in an RGB image. The resulting bounding box defines the region of interest for AAE, which determines the latent representation of the object's orientation. Using cosine-similarity, the most similar latent representation is selected from the codebook together with the corresponding orientation in 3D space. The object's translation can be calculated with the pinhole camera model of the calibrated camera, the image position obtained from the 2D object detector, and a projective distance estimation. The latter is determined by saving the bounding box diagonal of the rendered object image during codebook generation, and computing the ratio of this diagonal and the diagonal in the test image bounding box. The resulting 6D pose estimate can optionally be refined with perspective correction

and point-to-plane ICP refinement. The pipeline was evaluated on the publicly available T-LESS dataset [19] and outperformed all results on the 2018 BOP benchmark [20]. While the exact pose estimation error varies between different objects and scenarios, it was shown that a robust estimation with translational errors below 25 mm is possible.

A different encoder-decoder architecture, called EagerNet, is used in [21]. Analogous to AAE, a 2D object detector estimates a region of interest in the camera image, which the network uses as input. EagerNet follows a correspondence-based approach and is trained to map image pixels to model point coordinates on the surface of the target object. A PnP algorithm then uses these 2D-3D correspondences to calculate the object pose. The network is trained on rendered images of the object's 3D model, minimizing the loss between 3D surface point prediction and corresponding point in the rendered 3D model for every pixel. Simultaneously, the network learns to map each surface point to one of n region classes, which are determined by applying farthest-point sampling. This way, EagerNet can better differentiate between similar-looking points on symmetric objects. Additionally, the network learns to segment the object from its background by trying to predict if a pixel lies within the target object mask. This is important to decide whether a 2D-3D correspondence should be considered by the PnP algorithm. To account for occlusion and modeling errors, an additional output channel is added, trained to predict the error of the predicted model coordinates for each foreground pixel. Instead of using a fixed error threshold to determine if a 2D-3D correspondence is considered by the PnP algorithm, multiple predictions with different error thresholds are produced during runtime.

The predictions from EagerNet are refined using a modified version of the region-based object tracking algorithm proposed in [22]. The probabilistic tracker uses image statistics to differentiate between foreground and background, and aims to find an object pose that best explains this segmentation. In addition, it returns a probability metric of the most likely pose, which can be interpreted as the prediction confidence. The pose prediction with the highest probability over all error thresholds is used as the final prediction of the pose estimator.

EagerNet achieved state-of-the-art performance on the SPEED+ satellite pose estimation dataset [23] in the post-mortem SPEC competition, organized by the European Space Agency and Stanford University [21]. The dataset is particularly challenging for 6D pose estimation, as the algorithm has to work on images of symmetric objects with reflective surfaces under extreme lighting conditions. Furthermore, EagerNet requires a 3D model of the target object, which was not provided. Instead, the authors designed a custom model based on the annotated images in the dataset, showing that the algorithm is able to work on approximate 3D models. This was also proven in an ablation study on the TUD-L dataset [20], where EagerNet achieved similar pose estimation performance when trained on deformed versions of the same 3D model [21].

3

Background

This chapter introduces the theoretical foundations required for the proposed extension of Kernelized Guided Reinforcement Learning (KGRL). First, Kernelized Movement Primitives (KMP) and their constrained and null-space variants are presented. Subsequently, KGRL is introduced as a method that integrates these representations into reinforcement learning.

3.1 Movement Primitives

Movement primitives provide a structured representation of demonstrated trajectories and have become an important tool for robot learning from demonstration [15]. Instead of directly reproducing recorded motions, movement primitives encode demonstrations in a way that captures their essential characteristics while allowing controlled adaptation to new task conditions. This makes them suitable for manipulation tasks, where both generalization and stability are required [14].

In the context of this work, movement primitives based on kernelized representations are presented, which allow trajectories to be learned from demonstrations and adapted based on additional constraints arising over the course of a task. The following sections introduce Kernelized Movement Primitives (KMP) [7] and their extensions.

Kernelized Movement Primitives (KMP)

KMPs provide a flexible framework for learning trajectories from demonstrations while preserving the variability observed in the data [7]. Compared to classical movement primitive formulations [15] such as Dynamic Movement Primitives (DMPs) [14] or Probabilistic Movement Primitives (ProMPs) [24], KMPs offer several advantages. In particular, many traditional approaches rely on explicit basis functions, such as Gaussian basis functions, whose number and shape must be chosen manually [15] [7]. In contrast, KMP employs a kernel-based, non-parametric formulation that avoids the need for explicit basis function design and reduces the number of open parameters [7]. Furthermore, the kernel formulation allows KMPs to handle multidimensional inputs

more effectively and to incorporate additional constraints, such as via-points or target conditions.

The starting point for KMP is a set D of M demonstrations[2], where each demonstration consists of a sequence of N states with input $s \in \mathbb{R}^I$ and output $\eta \in \mathbb{R}^O$:

$$D = \{\{s_{n,m}, \eta_{n,m}\}_{n=1}^{N}\}_{m=1}^{M}$$

For time-driven trajectories, which are used in [7] and [8] and also considered for the rest of this thesis, the input is a time step t ($s_n = t_n$), and the output is the corresponding robot pose at that time step ($\eta_n = p_n$).

To extract a consistent representation from multiple demonstrations, KMP first constructs a probabilistic reference trajectory. This is achieved by fitting a Gaussian Mixture Model (GMM) with C Gaussian components to the joint distribution of inputs and outputs [25], resulting in the probabilistic policy:

$$P(s, \eta) \sim \sum_{c=1}^{C} p_c \mathcal{N}(\mu_c, \Sigma_c),$$

where, p_c , μ_c ,and Σ_c are the probability, mean, and variance of each Gaussian component. By applying Gaussian Mixture Regression (GMR) [26], a new mean $\hat{\mu}_n$ and covariance $\hat{\Sigma}_n$ for every time step n are computed, resulting in the probabilistic reference trajectory T_r:

$$T_r = \{\hat{\mu}_n, \hat{\Sigma}_n\}_{n=1}^{N}$$

For the construction of KMP, a parametric, probabilistic trajectory is defined:

$$\eta(s) \sim \mathcal{N}(\Theta(s)^T \mu_w, \Theta(s)^T \Sigma_w \Theta(s)),$$

where $\Theta(s) \in \mathbb{R}^{BO \times O}$ is a block matrix with a B-dimensional, manually defined basis function vector $\varphi(s)$ on its diagonal, and μ_w and Σ_w are optimized to fit the parametric trajectory to the probabilistic reference trajectory T_r. This is achieved by minimizing the Kullback–Leibler (KL) divergence between the two trajectory distributions. The problem is separated into a *mean minimization subproblem* and a *covariance minimization subproblem*, where the *mean minimization subproblem* is given as[3]:

$$\underset{\mu_w}{\operatorname{argmin}} \sum_{n=1}^{N} \frac{1}{2} \left(\Theta^\top(s_n)\mu_w - \hat{\mu}_n\right)^\top \hat{\Sigma}_n^{-1} \left(\Theta^\top(s_n)\mu_w - \hat{\mu}_n\right) + \frac{1}{2}\lambda\, \mu_w^\top \mu_w.$$

By kernelizing the basis function and solving the *mean minimization subproblem*, the expected value for the optimized parametric trajectory becomes:

$$\mathbb{E}(\eta(s)) = k^*(K + \lambda\Sigma)^{-1}\mu,$$

which is the predicted KMP trajectory. k^* and K are kernel matrices, $\mu = \left[\hat{\mu}_1^\top, \hat{\mu}_2^\top, \ldots, \hat{\mu}_N^\top\right]^\top$ and $\Sigma = \text{blockdiag}(\hat{\Sigma}_1, \hat{\Sigma}_2, \ldots, \hat{\Sigma}_N)$ are constructed from T_r, and λ is a regularization term.

[2] For comparability and consistency, the notation from [8] is adopted.

[3] Again, the notation from [8] is adopted, in which the factor '$\frac{1}{2}$' is not omitted.

For the definitions of k^* and K, as well as the detailed mathematical derivation, please refer to [7] and [8].

Similarly, the formulation for the covariance prediction is obtained by solving the *covariance minimization subproblem*. It is also described in [7] and will not be explained in detail here.

The predicted KMP output $\mathbb{E}(\eta(s))$ can be used as a reference trajectory, that controls a robot based on the input s. Considering the previous definitions of $s_n = t_n$ and $\eta_n = p_n$, the KMP would return a robot position based on the current time step. The resulting trajectory does not necessarily align precisely with the mean of the reference trajectory T_r. Nevertheless, it maintains smooth behavior while remaining within the regions covered by the demonstrations. The reason for this is that $\mathbb{E}(\eta(s))$ is determined through an optimization process. In regions where the reference covariance is small, deviations from the mean trajectory can lead to a large increase in KL-divergence, resulting in trajectories that closely follow the demonstrations. In contrast, in regions with larger covariance, deviations are penalized less strongly, allowing the trajectory to differ more from the mean while still remaining consistent with the demonstrated variability. The resulting trajectory can therefore be interpreted as the most consistent estimate with respect to both the demonstrated mean behavior and its associated uncertainty.

The adaptability of KMP arises from the fact that the optimization can be recomputed under new conditions, without the need to record new demonstrations. For instance, when additional constraints or via-points are introduced, the optimization can be repeated to obtain a new trajectory that satisfies these conditions while remaining consistent with the demonstrations.

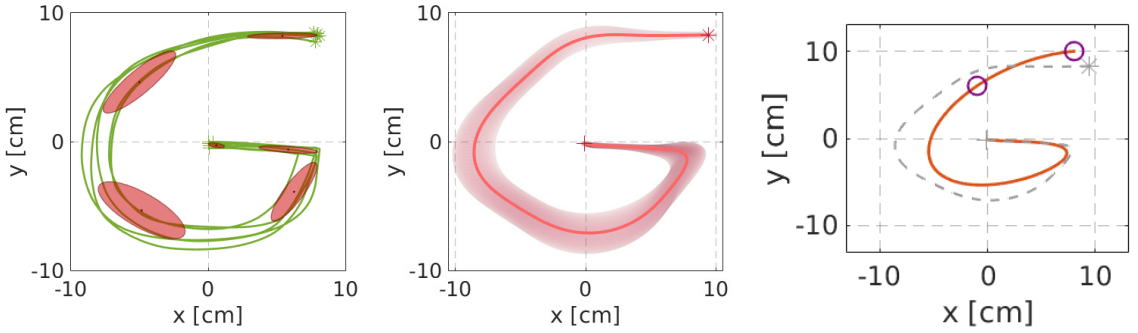


Figure 3.1: Generation of a KMP trajectory. (a) shows the demonstrated trajectories in green, a GMM was fitted to the demonstrations. The Gaussian components are represented by red ellipses. The symbols '*' and '+' indicate the start- and end-positions for the respective trajectories. (b) shows T_r obtained from GMR, with the mean trajectory as a red line and standard deviation as the shaded area. Finally, the KMP output is presented with a red line in (c), with two purple circles showing via-points that the KMP output has to go through. The mean reference trajectory is given in gray for comparison. All three images are taken from [7].

An example for the generation of a KMP trajectory can be seen in Figure 3.1. The simple 2D example was taken from [7]. First, $M = 5$ trajectories are demonstrated, and a GMM with $C = 6$ Gaussian components is fitted to the demonstrations (Figure 3.1a). Then, a reference trajectory T_r is obtained from GMR (Figure 3.1b), with mean (red line) and covariance. Finally, Figure 3.1c shows a generated KMP trajectory in red. Two via-points (purple circles) are defined after recording the demonstrations, which the trajectory has to target. Despite this modulation, the trajectory is smooth and follows the demonstrated behavior.

Linearly Constrained KMP (LC-KMP)

In [9], KMPs are extended with linear constraints, resulting in the LC-KMP formulation. Independent of the demonstrations, task-specific linear inequality constraints are incorporated that limit the robot's movements. This is achieved by extending the aforementioned minimization of KL divergence between kernel model and reference trajectory distribution, to obtain a *constrained mean minimization subproblem*[4]:

$$\underset{\mu_w}{\operatorname{argmin}} \sum_{n=1}^{N} \frac{1}{2} \left(\Theta^\top(s_n)\mu_w - \hat{\mu}_n\right)^\top \hat{\Sigma}_n^{-1} \left(\Theta^\top(s_n)\mu_w - \hat{\mu}_n\right) + \frac{1}{2}\lambda\, \mu_w^\top \mu_w,$$

$$\text{s.t. } g_{n,f}^\top \eta(s_n) \geq c_{n,f}, \forall f \in \{1,2,\dots,F\}, \forall n \in \{1,2,\dots,N\},$$

where $g_{n,f}$ and $c_{n,f}$ are manually defined constraints, and F is the number of constraints. By kernelizing the basis function again and introducing the Lagrange multiplier vector α [8] [9], this leads to the following solution of the optimization problem:

$$\mathbb{E}(\eta(s)) = k^*(K + \lambda\Sigma)^{-1}\mu + k^*(K + \lambda\Sigma)^{-1}\Sigma\bar{G}\alpha,$$

which is the LC-KMP output. The Matrix $\bar{G}$ includes the linear constraints on its diagonal, and the optimal Lagrange multiplier vector α is constructed by solving an additional optimization problem that is defined in [9]. The parameters k^*, K, μ, Σ, and λ are defined as before. For the mathematical derivation, please refer to [9]. Note that the first term of the NS-KMP formulation is identical to the KMP output.

This extension of the KMP formulation results in the constraints being integrated into the trajectory generation process, ensuring that the resulting motion satisfies the desired conditions while remaining as close as possible to the demonstrated behavior. The 2D example in Figure 3.2, which was taken from [9], visualizes this. The LC-KMP was constructed from the same demonstrations that are given in Figure 3.1. Upper and lower bounds are defined in the x-direction for the robot, and a lower bound in the y-direction. The resulting LC-KMP trajectory follows the via-points, adheres to the linear constraints, and stays close to the demonstrations given in Figure 3.1.

[4] Again, the notation from [8] is adopted, in which the factor '$\frac{1}{2}$' is not omitted (as opposed to [9]).

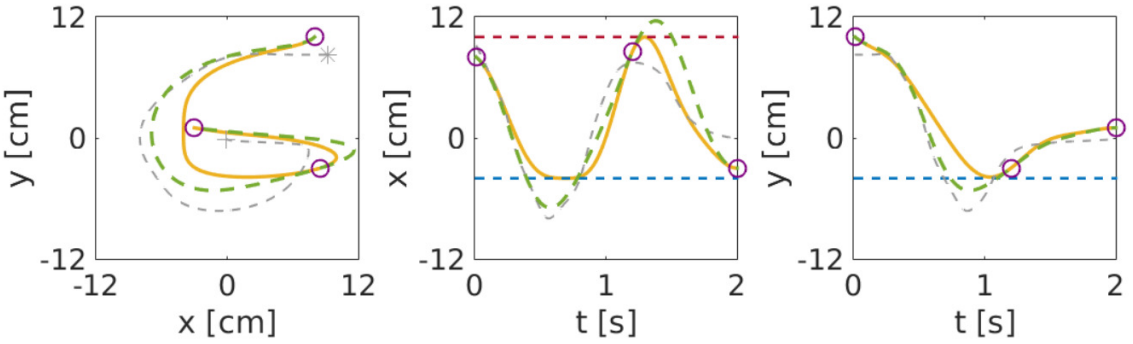


Figure 3.2: LC-KMP trajectory with via-points and linear constraints. The via-points are shown as purple circles. The gray dashed line shows the reference trajectory T_r, the green dashed line shows the regular KMP output. The yellow line shows the LC-KMP trajectory, which adheres to the linear constraints, visualized by the red and blue dashed lines. The trajectories are shown in XY-plane (left) and as x- and y-positions over time (center and right). The images were taken from [9].

Null-Space KMP (NS-KMP)

NS-KMP, introduced in [27], extends the KMP formulation to allow modulation of a generated trajectory in order to accomplish secondary objectives. This is achieved by incorporating the concept of a null-space projector [28] directly into the trajectory generation process. The original KMP *mean minimization subproblem* is extended with the cost term $\frac{1}{2}\beta(\mu_w - \hat{\mu}_w)^\top(\mu_w - \hat{\mu}_w)$, which keeps the solution closed to a desired one $\hat{\mu}_w$. This leads to a new optimization problem[5]:

$$\begin{aligned}\underset{\mu_w}{\operatorname{argmin}} \sum_{n=1}^{N} \frac{1}{2} \left(\Theta^\top(s_n)\mu_w - \hat{\mu}_n\right)^\top \hat{\Sigma}_n^{-1} \left(\Theta^\top(s_n)\mu_w - \hat{\mu}_n\right) \\ + \frac{1}{2}\lambda\, \mu_w^\top \mu_w + \frac{1}{2}\beta(\mu_w - \hat{\mu}_w)^\top(\mu_w - \hat{\mu}_w).\end{aligned}$$

By solving this problem and by kernelizing the basis function and null-space solution, the output for NS-KMP is:

$$\mathbb{E}(\eta(s)) = k^* A\mu + \frac{\beta}{\gamma}(\hat{k}^* - k^* A\hat{K})\underline{K}^{-1}\hat{\xi},$$

where A is defined as:

$$A = (K + \lambda\Sigma)^{-1}.$$

Again, k^*, K, μ, Σ, and λ are defined as for the KMP formulation above. Additionally, γ and β are both hyperparameters, with $\gamma = \lambda + \beta$. The matrices $\hat{k}^*$, $\hat{K}$ and $\underline{K}$ are constructed from the basis function. For their definition and the detailed derivation, please refer to [27] and [8]. The tunable parameter $\hat{\xi}$ is called the null-space reference. As in the formulation of LC-KMP, the first term for NS-KMP is equivalent to the KMP output.

[5] Again, the notation from [8] is adopted, in which the factors '$\frac{1}{2}$','β', and 'λ' are not omitted (as opposed to [27]).

In robotics, null-space projectors are generally designed to ensure that secondary tasks do not interfere with a primary task [28]. However, the projection used for NS-KMP is constructed from demonstration data and embedded into the probabilistic formulation. As a result, it does not strictly preserve the original trajectory, and is therefore referred to as a *soft null-space projector* [27]. Instead of enforcing a strict separation between task and null-space, it allows the projector to deform the original KMP trajectory using the null-space reference $\hat{\xi}$. The strength of this deformation is determined by the variability observed in the demonstrations. Regions with low variability in the demonstrations constrain the trajectory strongly, limiting the effect of the null-space modulation. In contrast, regions with higher variability allow greater deviation. This property is desirable, as it naturally prioritizes the demonstrated behavior in well-defined regions of the task, while allowing flexibility where the demonstrations exhibit uncertainty. NS-KMP enables efficient and adaptive modulation of trajectories without requiring re-optimization of the underlying movement primitive.

Linearly Constrained Null-Space KMP (LC-NS-KMP)

In [8], LC-KMP and NS-KMP are combined, leading to the formulation of LC-NS-KMP. Applying the linear constraints from LC-KMP to the NS-KMP optimization problem yields the following:

$$\begin{aligned}\underset{\mu_w}{\operatorname{argmin}} \sum_{n=1}^{N} \frac{1}{2} \left(\Theta^\top(s_n)\mu_w - \hat{\mu}_n\right)^\top \hat{\Sigma}_n^{-1} \left(\Theta^\top(s_n)\mu_w - \hat{\mu}_n\right) \\ + \frac{1}{2}\lambda\, \mu_w^\top \mu_w + \frac{1}{2}\beta(\mu_w - \hat{\mu}_w)^\top(\mu_w - \hat{\mu}_w), \\ \text{s.t. } g_{n,f}^\top \eta(s_n) \geq c_{n,f}, \forall f \in \{1,2,\dots,F\}, \forall n \in \{1,2,\dots,N\}.\end{aligned}$$

By solving the optimization problem as described above, this leads to the LC-NS-KMP formulation:

$$\mathbb{E}(\eta(s)) = \underbrace{k^* A \mu}_{\tau_1} + \underbrace{k^* A \Sigma \bar{G} \alpha}_{\tau_2} + \underbrace{\frac{\beta}{\gamma}(\hat{k}^* - k^* A \hat{K}) \underline{K}^{-1} \xi}_{\tau_3},$$

with

$$A = (K + \lambda\Sigma)^{-1}.$$

Following the notation from [8], ξ is referred to as the null-space action. Note how the first term (τ_1) is the KMP output, τ_2 is the term that extends the KMP with linear constraints, and τ_3 is the term that integrates the soft null-space projector.

This results in a representation where trajectories can be constrained manually while still allowing flexibility in unconstrained areas. The nominal KMP trajectory can be modulated by applying a null-space action ξ at particular time steps. The soft null-space projector scales the effect of the modulation based on the covariance from the probabilistic reference trajectory, ensuring that the modified trajectory preserves the characteristics of the demonstrations.

Figure 3.3 shows the generation of a modulated LC-NS-KMP trajectory on a simple 2D example. The example and the Figure were taken from [8]. First, five demonstrations are recorded that map each time step t to a position. A GMM with five Gaussians is fitted to these demonstrations (Figure 3.3-A1). Through GMR, a probabilistic reference trajectory is obtained, with which the LC-NS-KMP can be constructed. Applying different values for the null-space action ξ at time step $t = 3.2$ s results in smooth modulated trajectories, which deviate from the mean KMP trajectory while staying similar to the demonstrations (Figure 3.3-A2). The magnitude and direction of the deviation does depend on the value of ξ, but more importantly, on the variations in the demonstrations. In Figure 3.3-A3, the effect of sampling ξ from a normal distribution at each time step is shown. Smoothness is not preserved, but the resulting trajectory points still follow the demonstrated motion. Additionally, the manually defined linear constraints, shown by the red dashed rectangle, are never violated.

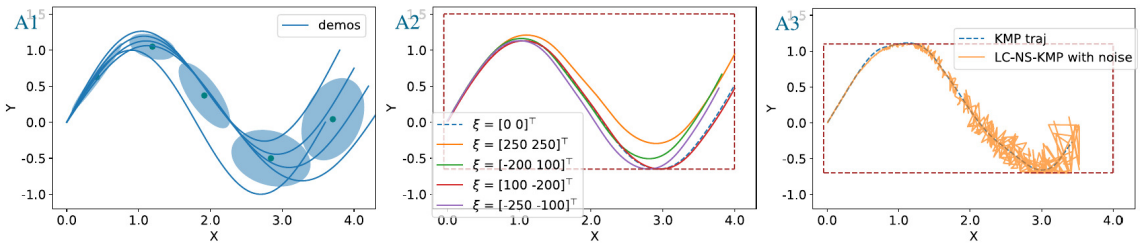


Figure 3.3: Generation of modulated LC-NS-KMP trajectories in a 2D example. A1: GMM fitted to demonstrated trajectories. A2: Effect of different null-space actions ξ applied at the same time step. A3: Random sampling of new values for ξ at every time step. The modulated LC-NS-KMP trajectories preserve the characteristics of the demonstrations while never violating the constraints (indicated by the red dashed rectangle). The Figure was taken from [8].

3.2 Kernelized Guided Reinforcement Learning (KGRL)

KGRL [8] extends the LC-NS-KMP framework described above by integrating reinforcement learning (RL) to generate the null-space actions ξ. Instead of manually defining null-space actions for specific time steps before execution, a reactive policy is learned that generates values for ξ during runtime, depending on the observations of the RL agent. Based on the input s (for example, the current time step for $s_n = t_n$) and an arbitrary state vector q (for example, the robot state), the output of KGRL is defined as follows:

$$\mathbb{E}(\eta(s)) = k^* A\mu + k^* A\Sigma\bar{G}\alpha + \frac{\beta}{\gamma}(\hat{k}^* - k^* A\hat{K})\pi(\xi|q).$$

Essentially, the RL policy $\pi(\xi|q)$ generates null-space actions ξ for the LC-NS-KMP, using observations q. The input q for the RL policy can be the same as the state s, but can also be different. While s has to be the same type of input that was used for the recording of demonstrations (for example, time), q can be any observable state (for example, from robot

sensors). Under the assumption from earlier that $\eta_n = p_n$, the output $\eta_n(s)$ is a target trajectory for the robot. This trajectory can be interpreted as a nominal LC-KMP trajectory that is modulated using the learned null-space action ξ.

The goal for the RL policy $\pi(\xi|q)$ is to optimize the trajectory generated by LC-NS-KMP with respect to a user-defined reward function. This reward function can be sparse and only consider whether a single task was solved or not, but it can also include secondary rewards to account for additional objectives or optimization metrics.

Using an LC-NS-KMP for trajectory generation restricts the RL exploration space to variations of the demonstrated trajectories. As previously discussed, and visualized in Figure 3.3, the covariance obtained from demonstrations determines how strongly the trajectory is constrained at each point in time. Regions with low covariance correspond to high confidence and restrict exploration, while regions with higher covariance allow greater flexibility. In this way, the demonstrations define the nominal trajectory and shape the admissible exploration space. Assuming that the user gives meaningful demonstrations and defines potential linear constraints, KGRL reduces the risk of unsafe exploration and improves sample efficiency. In fact, KGRL enables efficient learning even in real-world systems, directly on the robot. Without requiring a simulation-based pretraining phase, the sim-to-real gap is avoided that commonly arises when transferring policies from simulation to the real world [8].

In [8], KGRL was evaluated in a simulation setting and on a robot in the real world. In simulation, the task for the robot is to navigate a partially constrained 2D environment to reach a goal. In order to do this, it has to pass through a narrow passage and avoid an obstacle. Demonstrations are provided that describe the behavior of moving through the passage to reach the goal, but without considering the constrained region or the obstacle. KGRL was compared to a simple residual policy, which modulates a KMP trajectory directly, without the use of a soft null-space projector. The residual policy was also extended to include the same linear constraints as KGRL, which were set to keep the robot out of the restricted zone in the environment. In contrast to both residual policies (with and without linear constraints), KGRL managed to solve the task reliably, quickly achieving a success rate of 1. It also learned to avoid the obstacle in under 30 episodes while needing less exploration than the residual approaches.

In the real world, KGRL was evaluated on the task of inserting a BNC connector with a robotic arm [8]. The task was selected from the NIST assembly benchmark [1], and is especially challenging because of the tight tolerances of the connector, as well as its sensitive twist-to-lock mechanism. Five demonstrations were provided to the robot, in which the last frame of each demonstration was assumed to be the target frame (the end-effector pose when the connector is locked). Due to dynamic and kinematic uncertainties, a simple KMP that was constructed from the demonstrations was not able to solve the task reliably. The reward function of KGRL was configured to return a sparse reward for successful task completion. Additionally, a secondary negative reward was included, scaled by the absolute value of measurements recorded by the robot's force-torque sensors. This way, a secondary objective of minimizing contact forces was specified. KGRL managed to achieve a success rate of 1 in under 45 episodes, while simultaneously optimizing according to the secondary objective.

Despite these advances, KGRL has a major limitation, which is also discussed by the authors in [8]. It assumes that a suitable exploration strategy can be inferred from the demonstrations. For many scenarios, especially in dynamic environments, this might not be the case. KGRL does not readily adapt when introducing variance into the system that was not present during the demonstrations. It could even be less effective than a simple residual policy when it constrains the exploration space to such a degree that the task cannot be executed. For example, consider the exploration behavior shown in Figure 3.3. Close to the point [0,0] in the 2D grid, the robot is restricted to barely deviate from the mean KMP trajectory, independent of the null-space action ζ. Introducing uncertainty into the environment that is not present in the demonstrations could lead KGRL to fail. The authors suggest that in such a scenario, manual tuning of the covariance matrix used for the construction of LC-NS-KMP could be a potential mitigation strategy. This thesis explores the idea of extending KGRL in such a way by modulating the covariance using external sources, particularly perception.

4

System Setup

This thesis was conducted at the Institute of Robotics and Mechatronics at the German Aerospace Center (DLR). As part of this work, a range of hardware and software systems were utilized, some of which are developed and maintained internally at DLR and may not be widely known outside the DLR ecosystem. This chapter provides a comprehensive overview of the systems involved to introduce the reader to specialized tools and technologies, especially those that are not publicly accessible. Furthermore, the chapter describes how the robot system was set up, which was used for real-world connector-insertion experiments. In this context, the calibration procedure for the camera system is also presented.

4.1 Software Components

DLR Links and Nodes (LN)

Links and Nodes is middleware that is used at DLR to manage flexible, distributed systems and to communicate between different processes [29]. It is comparable to the Robot Operating System (ROS). A user can define nodes that exchange predefined messages by publishing or subscribing to a topic, or by calling or providing services. Nodes, messages, topics, and services can be managed with a configurable graphical user interface, the LN manager. LN is real-time compatible and supports multiple programming languages (C, C++, Python2, Python3, and Matlab/Simulink). For use outside DLR, LN can be built using the repository found under [30].

DLR Portable Computer Vision Pipeline (PCVP)

The Portable Computer Vision Pipeline is an internal DLR framework that is not publicly available. PCVP manages the lifecycle of different computer vision modules and establishes communication between them via standardized message types. In contrast to LN, the message flow has to be specified in a configuration file before runtime. Messages can be synchronized between multiple modules, which is useful for setups in which more than one module has to work on the same data. Modules can be created using either Python or C++ and have to be specified with inputs and outputs following the PCVP

message definition. At DLR, several useful computer vision applications are available as PCVP modules. Some of these, namely the YOLO detector and the EagerNet module, were used for this work and will be introduced in Chapter 5.1. PCVP modules can also provide LN nodes, enabling the integration of a PCVP pipeline into an LN manager and facilitating communication with non-PCVP software.

DLR CalDe and DLR CalLab

DLR CalDe and DLR CalLab are software tools developed in 2005 at the DLR Institute of Robotics and Mechatronics [31]. Together, they form a toolbox for camera calibration using a checkerboard pattern (see Figure 4.7). The calibration process is divided into two separate steps: feature detection is performed in DLR CalDe, while parameter estimation is handled in DLR CalLab. An old version of the software is publicly available [31].

The calibration is based on correspondences between known points on the checkerboard pattern and their positions in the captured images. These correspondences are stored in *.pts* files, which contain pairs of 3D coordinates in the pattern coordinate system and their corresponding pixel coordinates. To generate the correspondences, the user first captures several camera images of a checkerboard pattern from different angles. The pattern should be created with DLR CalDe, as it has to include three circular markers in its center in addition to the black and white squares. These markers define the orientation of the pattern's coordinate system. DLR CalDe is then used to detect the corner points of the checkerboard in the images. This can be done either automatically or semi-automatically. In semi-automatic mode, the user selects the three central circles to provide an initial estimate of the pattern orientation. This improves the robustness of the detection in difficult lighting conditions and reduces computation time, especially for high-resolution images. An example of the detection result is shown in Figure 4.7. Based on the known geometry of the pattern, the detected pixel positions are assigned to their corresponding coordinates in the pattern frame and saved in *.pts* files.

These files are then used in DLR CalLab to estimate the intrinsic and extrinsic camera parameters. Intrinsic parameters describe how points in the camera coordinate system are projected onto the image and depend on the internal properties of the camera, such as lens and sensor distortions. Extrinsic parameters describe the position and orientation of the camera relative to a reference frame [32] [33] [34] [35]. This is important, for example, in hand-eye calibration, where a camera is mounted onto a robot [31]. In this case, the extrinsic parameters define the transformation between the camera and the robot's Tool Center Point (TCP). If the transformation between the robot base and the TCP is known, for instance, from forward kinematics, the camera position can be expressed relative to the robot base. To include this information in the calibration, the transformation between the TCP and the robot base must be recorded for each image and stored in a *.coords* file. This file can be loaded into DLR CalDe, which adds the transformation data to the corresponding *.pts* files.

For the calibration itself, the images and the associated *.pts* files are loaded into DLR CalLab. The detected corner points are displayed and can be manually corrected by removing obvious outliers. This helps to improve the accuracy of the parameter estimation. The calibration is performed in two steps: first, the intrinsic parameters are estimated,

and in an optional second step, the extrinsic parameters are calculated. The results are stored in a *.cal* file, which contains the intrinsic parameters and, if computed, also the extrinsic parameters. The intrinsic parameters consist of a camera matrix as defined in [35] and multiple distortion parameters, depending on which lens distortion model the user chooses [31]. Depending on user specification, CalLab can consider radial distortion, translational distortion, and thin prism distortion [33]. The extrinsic parameters consist of a rotation matrix and a translation vector, describing the transformation between camera and reference frame.

4.2 Hardware Components

The SARA Robot

The Safe Autonomous Robotic Assistant (SARA), shown in Figure 4.1, is a lightweight robotic manipulator developed at the DLR Institute of Robotics and Mechatronics [3] [36]. The system features seven degrees of freedom and is equipped with torque sensors in each joint, in addition to force-torque sensors located at both the base and the wrist. This sensor setup enables uncoupled force-torque measurements in Cartesian space that are independent of the joint configuration and not affected by singularities [4] [36]. Multiple simultaneous external contacts can be robustly identified and localized in real time, with a resolution finer than 0.1 N. This makes the robot especially well-suited for contact-rich manipulation tasks.

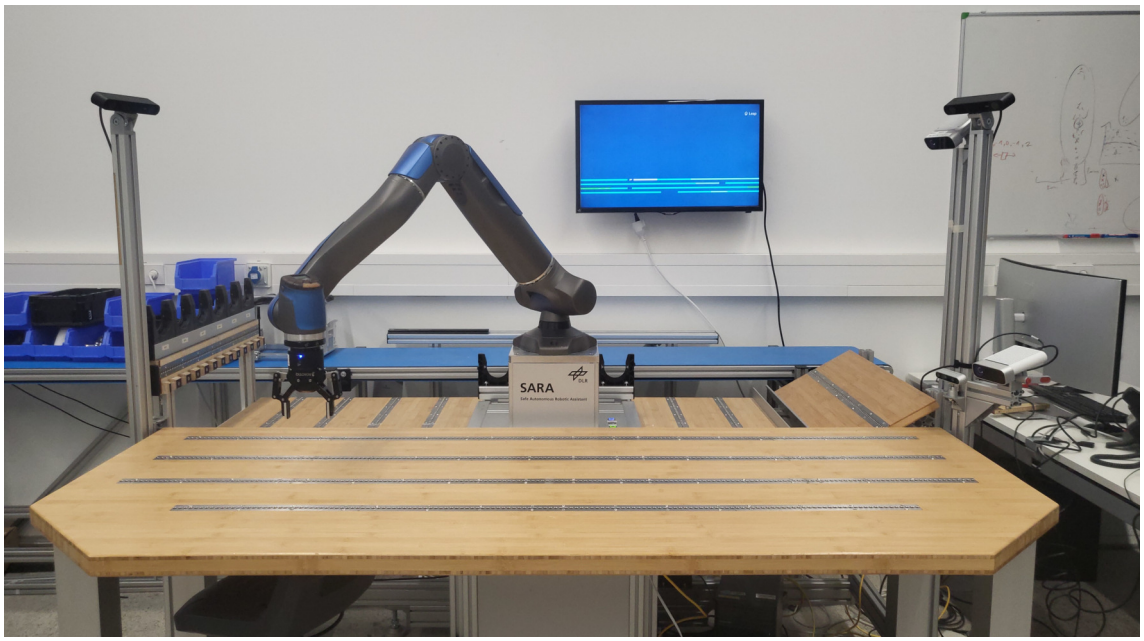


Figure 4.1: The SARA robot mounted to the workbench. The robot is configured with a two-finger gripper tool. Apart from the Microsoft Azure Kinect and the Intel RealSense cameras that were used for this thesis (right side of the workbench), the workbench is configured with three additional cameras that capture the workbench from above.

SARA was used to evaluate the original KGRL algorithm in [8] and therefore chosen for the experiments in this thesis. As shown in Figure 4.1, the robot is mounted to a workbench featuring metallic rails with regularly spaced holes. This setup allows objects to be securely positioned within the workspace using dedicated clamping mechanisms. The rails have a distance of 16 cm between each other, the offset for the holes on each rail is 1 cm. For this thesis, SARA was configured with a two-finger gripper tool. Configuration and programming of the SARA robot can be performed either via a graphical user interface (GUI) or through the DLR-internal Python library *pysara*.

An important operational mode that SARA has is gravity compensation, in which the control system counteracts gravitational forces acting on the joints. In this mode, the robot can be freely guided by a user into any desired configuration. This functionality was utilized in this work to record demonstration trajectories. The user manually manipulates the end-effector to perform the desired task, while the system simultaneously logs the corresponding end-effector poses and force-torque data over time. In addition, SARA provides precise collision detection, which is triggered when unexpected external forces exceed a specified threshold [4].

Microsoft Azure Kinect Developer Kit

The Microsoft Azure Kinect Developer Kit is a depth-sensing camera system that combines a 12-megapixel RGB camera with a one-megapixel time-of-flight depth sensor [37]. The maximum supported resolution for capturing RGB images is 4096x3072 pixels. Access to the image and sensor data is provided through the Python library *pyk4a*[6].

Intel RealSense D435

The Intel RealSense D435 is a stereo depth camera that integrates an RGB sensor with a pair of infrared stereo cameras used for depth estimation. The system provides an RGB image stream with a resolution of up to 1920x1080 pixels and a depth image stream with a resolution of up to 1280×720 pixels [38]. Access to both RGB images and depth data is provided via the Intel RealSense SDK 2.0, which can be used in Python through the *pyrealsense2* library[7].

4.3 Experiment Setup

Although the proposed use case for evaluating EU-KGRL is to learn robust connector-insertion, the implemented vision system does not directly estimate the pose of the connector port itself. Instead, the pose estimation (described in Chapter 5.1) is performed on the overall device, from which the connector port pose is derived via a known transformation.

This design choice is motivated by several practical considerations. On the SARA robot that was used for this work, a functional wrist-mounted camera was unavailable, making

[6] https://github.com/etiennedub/pyk4a/ (visited on 04/30/2026)
[7] https://pypi.org/project/pyrealsense2/ (visited on 04/30/2026)

it impossible to follow the approach used in [2] to obtain close-range, high-resolution observations of the port. Instead, cameras mounted onto the side of the workbench were used, which provide images of the entire robot workspace. Estimating the pose of the device instead of the port allows for the usage of this static camera setup, even in situations where the connector port is not facing the camera. The setup allows operation within a larger workspace without requiring precise prior localization, which would be necessary for estimating the connector port pose directly [2]. Furthermore, detecting the device as a whole reduces ambiguity and instability that can arise when directly targeting small, textureless, or reflective connector components, although this may depend on the device in question. As robotic systems repeatedly interact with the same objects in typical industrial scenarios, the pose estimator only has to be configured to consider a limited set of objects that are known in advance.

In the initial robot setup, a Microsoft Azure Kinect camera and two Stereolabs ZED2[8] cameras were available, and configured to observe the workspace from above (see Figure 4.1). First experiments with these cameras showed insufficient pose estimation accuracy, which was suspected to be caused by the large distance between the cameras and the target object. Therefore, the setup was extended with cameras that were mounted closer to the workbench surface. As shown in Figure 4.1 and Figure 4.2, an Intel RealSense camera and a Microsoft Azure Kinect camera were attached to the right side of the workbench. The choice to install both cameras was made to test the system under different hardware conditions.

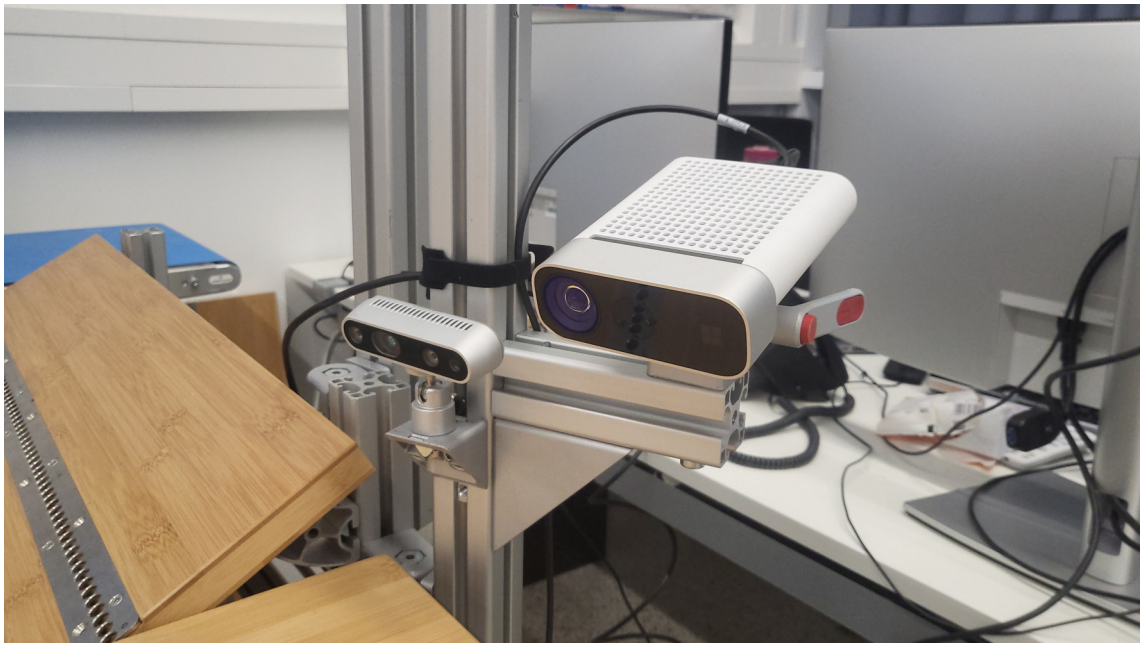


Figure 4.2: The Intel RealSense camera (left) and Microsoft Azure Kinect camera (right), that were used for the pose estimator in this work. They were attached close to the workbench surface, to be able to observe the target object from a close distance.

[8] https://www.stereolabs.com/en-de/products/zed-2 (visited on 04/30/2026)

For experimental evaluation, a Raspberry Pi 5 single-board computer [39] was selected as the target device. The reasons for this are its widespread availability, standardized form factor, and the presence of multiple commonly used connectors, including an Ethernet port. Additionally, the manufacturer provides an accurate 3D model [40], which was relevant for the configuration of the implemented pose estimation pipeline (see Chapter 5.1).

In order for the robot to be able to insert the connector without moving the device, the device must be fixed to the workbench. For this, the mounting mechanism shown in Figure 4.3 was designed. The Raspberry Pi is screwed onto a plastic plate, which is secured to an aluminum beam. The 3D-printed plate was modeled using a CAD software, while considering the mechanical dimensions of the Raspberry Pi specified by the manufacturer [40]. The aluminum beam connects the plate to the clamping mechanism for the workbench rails at a 90° angle. This angle makes it easier for the robot to access the Ethernet port of the device. Using the clamp adapter, the Raspberry Pi can be mounted to one of the workbench rails, aligned with any of the regularly spaced holes.

The plastic adapter plate is wider than the circuit board of the Raspberry Pi. This design choice was made deliberately to ensure that the vision system would always observe the device in front of the same background, providing similar conditions for the pose estimation regardless of the workspace configuration. For testing purposes, the plate was printed in different colors (white, black, yellow, and blue), but no effect in pose estimation accuracy could be observed. The experiments presented in this thesis were therefore conducted using a white plastic plate.

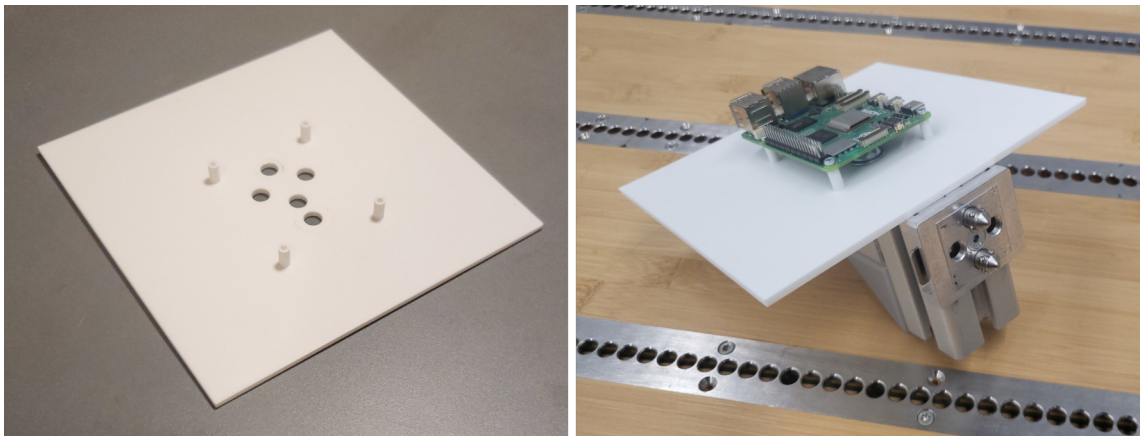


Figure 4.3: The mounting mechanism for the Raspberry Pi. The 3D-printed plate (left) is fixed to a metal beam, which is connected to the clamping mechanism (right) at a 90° angle.

The task of handling a cable was not considered in this thesis. In order to limit the problem to connector-insertion only, the end of a standard Ethernet cable was cut off and used for the experiments. This connector was placed into a 3D-printed holder, from which the robot could pick it up (see Figure 4.4). The holder was designed for this thesis in a CAD software to closely match the mechanical dimensons of the Ethernet connector obtained from measurements with a ruler, and attached to a rail mounting clamp. This

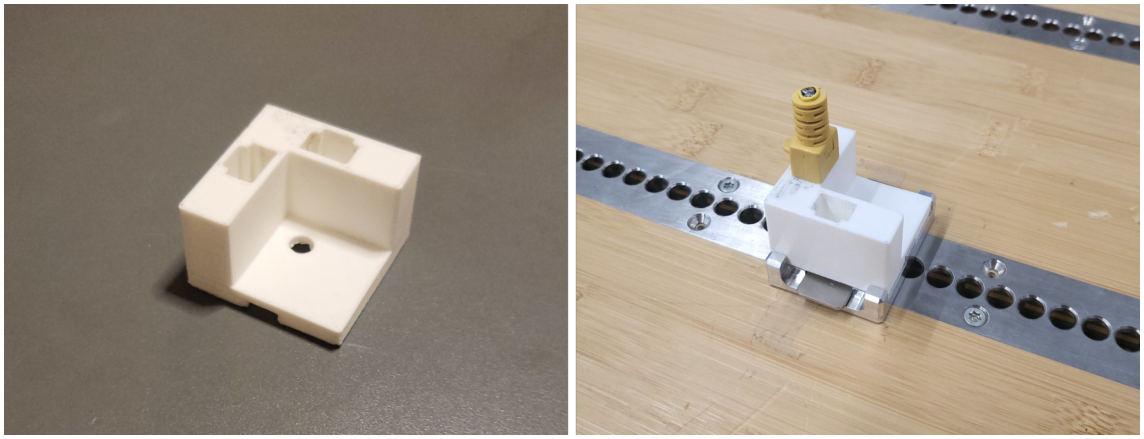


Figure 4.4: The 3D-printed holder for the Ethernet connector (left). Attached to the clamping mechanism (right), it can be fixed to the workbench, so that the robot can pick up the connector.

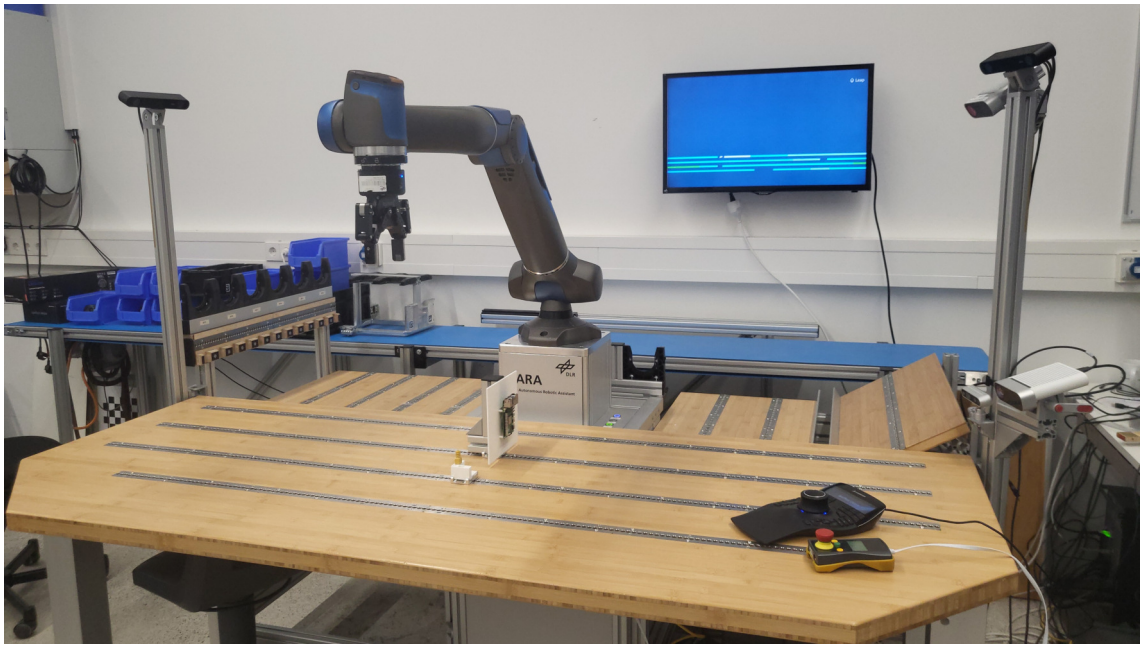


Figure 4.5: The final experiment setup. The Intel RealSense and Microsoft Azure Kinect cameras are mounted on the right side of the robot workbench. The Raspberry Pi and the Ethernet connector holder are attached to the metal rails using the clamping mechanisms. For the user, an external input device and an emergency stop button are placed on the workbench. A screen is mounted behind the robot to display relevant information during the training process.

way, it could be securely fixed to the workbench, ensuring that the robot could always pick up the connector from the same pose. For this thesis, it was assumed that the robot is provided with the exact point from which it must pick up the connector and only needs to estimate the pose of the Raspberry Pi. Depending on the use case, the setup could also be extended to determine the initial connector pose using a pose estimator.

In addition, an external input device was placed on the workbench, which allows the user to start and stop the RL training process, and to optionally give manual rewards to the RL agent during training. The training progress could be observed on a screen that was mounted to the wall behind the robot. In addition to the collision detection feature, which is active throughout the experiment, an emergency stop button was placed on the workbench for increased safety. The final experiment setup is shown in Figure 4.5.

4.4 Camera Calibration

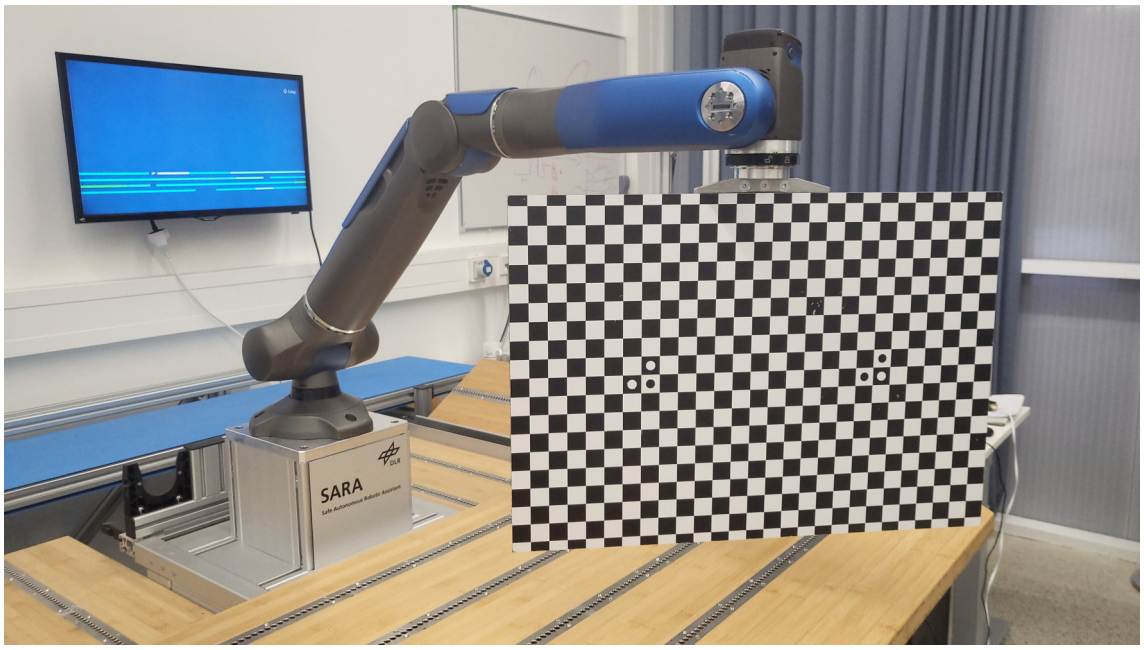


Figure 4.6: The SARA robot configured with the checkerboard pattern tool. The pose of the calibration pattern can be obtained using the *pysara* library.

In order to implement a pose estimator that produces accurate predictions in the robot frame, the newly attached Intel RealSense and Microsoft Azure Kinect cameras have to be calibrated. In the context of computer vision, this means determining the intrinsic and extrinsic camera parameters for the pinhole camera model [32] [33] [34] [35]. DLR CalDe and DLR CalLab (see Section 4.1) were chosen for this because they give the user a high level of control over the calibration process. With these tools, point correspondences can be checked and modified before the actual calibration, and outliers can be removed manually. In addition, a calibration pattern that was generated with DLR CalDe was already available. The pattern, which is printed on a plate, could be mounted to the SARA

robot, as shown in Figure 4.6. This setup allows the pattern to be positioned precisely in predefined poses.

An internal camera calibration procedure for such a setup was already available, and extended for this work to be compatible with different camera models. The underlying idea is the opposite of hand-eye calibration. Instead of mounting a camera to a robot to capture a calibration pattern from various perspectives, the calibration pattern is mounted to the robot so that a fixed camera can inspect it from different angles. The robot moves to different poses, and the camera takes images of the pattern in the different configurations. For each image, the current transformation between the robot base and the calibration pattern is obtained using the *pysara* library and stored in a *.JSON* file. A separate script is used to convert the *.JSON* file into individual *.coords* files, saving the transformation for each image. In total, 14 calibration images were recorded with different pattern poses, in which the pattern faces the camera.

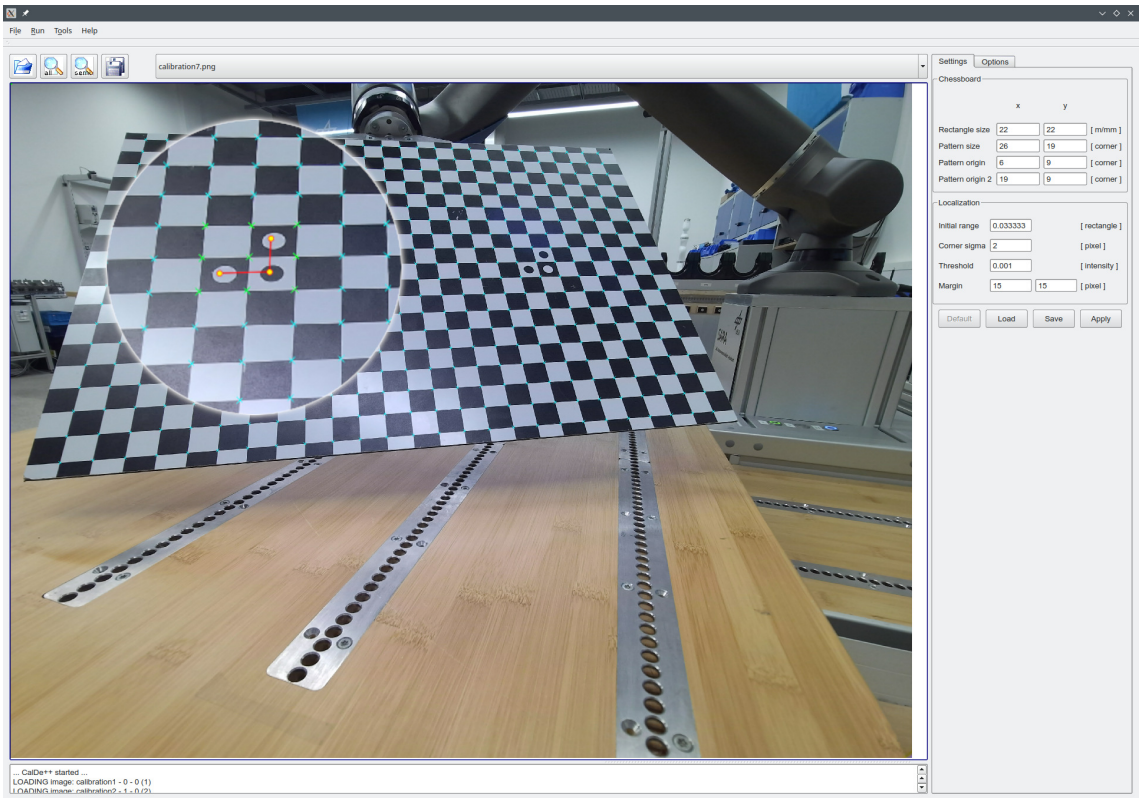


Figure 4.7: A screenshot of DLR CalDe after corner point detection on the checkerboard pattern. The detected corner points are marked with turquoise crosses. The central circles selected by the user as part of the semi-automatic detection process are highlighted in yellow. The calibration image was recorded using the Microsoft Azure Kinect camera.

As described in Section 4.1, the recorded images together with the *.coords* files were loaded into DLR CalDe to detect the corners of the checkerboard pattern. Because the calibration images were recorded with a high camera resolution, the semi-automatic mode was chosen. The detection result for one calibration image is shown in Figure 4.7. The *.pts* files exported from DLR CalDe were then loaded into DLR CalLab to estimate

the intrinsic and extrinsic camera parameters. The result was written to a *.cal* file. This procedure calibrated the camera in robot base frame.

The described calibration procedure was carried out for both cameras. They were configured to record the calibration images with the highest supported resolution (1920x1080 pixels for the Intel RealSense and 4096x3072 pixels for the Microsoft Azure Kinect). In the case of the Microsoft Azure Kinect camera, the manufacturer already provides accurate intrinsic parameters, including the camera matrix and distortion coefficients. They can be accessed using the camera's software development kit [41] and do not have to be determined again. For this reason, DLR CalLab was configured to include these intrinsic camera parameters for calibration. This results in DLR CalLab only estimating the extrinsic parameters for the camera and copying the provided intrinsic parameters to the calibration file without any changes. For the Intel RealSense camera, the intrinsic parameters were estimated using DLR CalLab, as the RealSense SDK does not provide precalibrated distortion coefficients[9].

[9] As discussed in https://github.com/realsenseai/librealsense/issues/1430 (visited on 04/30/2026)

5

Methodology

In this thesis, KGRL is extended by manually tuning the covariance that is used for the LC-NS-KMP formulation, in order to consider uncertainty that is not encoded by the user demonstrations. In the proposed real-world use case of connector-insertion, where the target pose is estimated using a camera, the uncertainty stems from the fact that pose estimates are generally imperfect. The uncertainty does not change over the course of the robot trajectory, as the camera system is static, and the target object is tightly secured to the workbench. Therefore, the target pose only needs to be estimated once before the robot starts to execute its trajectory. Given this (potentially inaccurate) pose estimate and an associated covariance that properly describes the distribution of pose estimation errors, an RL agent is trained to correct the nominal KMP trajectory to solve the insertion task based on interactions with its environment.

This chapter describes the development of the components that are necessary to realize the described setup. First, the implementation of a camera pose estimator is presented, which also provides information about estimation covariance. After that, the modifications to the virtual environment from [8] are described that were necessary to analyze the proposed extension of KGRL in a simulation that considers target pose uncertainty. Finally, the chapter presents how KGRL is extended to consider the target estimate and the associated covariance from either the pose estimator or the simulated environment, leading to the formulation of EU-KGRL.

5.1 Camera Pose Estimation Pipeline

The implemented pose estimation pipeline is based on EagerNet [21], which was introduced in Chapter 2. While originally designed to estimate the pose of satellites, EagerNet is also particularly well-suited for industrial applications. It demonstrates strong performance on objects with challenging visual characteristics, such as reflective surfaces and geometric symmetries, which are commonly encountered in manufacturing scenarios. Additionally, EagerNet only requires a 3D model of the target object for training, eliminating the need for manual data annotation. This significantly reduces the effort required to deploy the system, as 3D models are typically available in industrial contexts [42]. Furthermore, the method is robust to inaccuracies in the provided 3D

geometry [21], enabling effective operation even when only approximate models are available. In such cases, users can generate models through 3D scanning or simplified reconstruction using 3D modeling software, while still achieving reliable pose estimates. For the experiment setup used in this thesis, this was not necessary, as an accurate 3D model was available [40].

The pose estimation pipeline was implemented using the Portable Computer Vision Pipeline (PCVP) that was introduced in Chapter 4.1. Figure 5.1 shows a schematic representation of the pose estimation pipeline. Apart from the Camera Frame Provider and the Calibration File Provider, it can be separated into four core modules: the Undistortion module, the YOLO detector, the EagerNet module, and the Pose Refiner. The implementation, as well as the data generation and training procedure that was required for some of these modules, will be described in the following section.

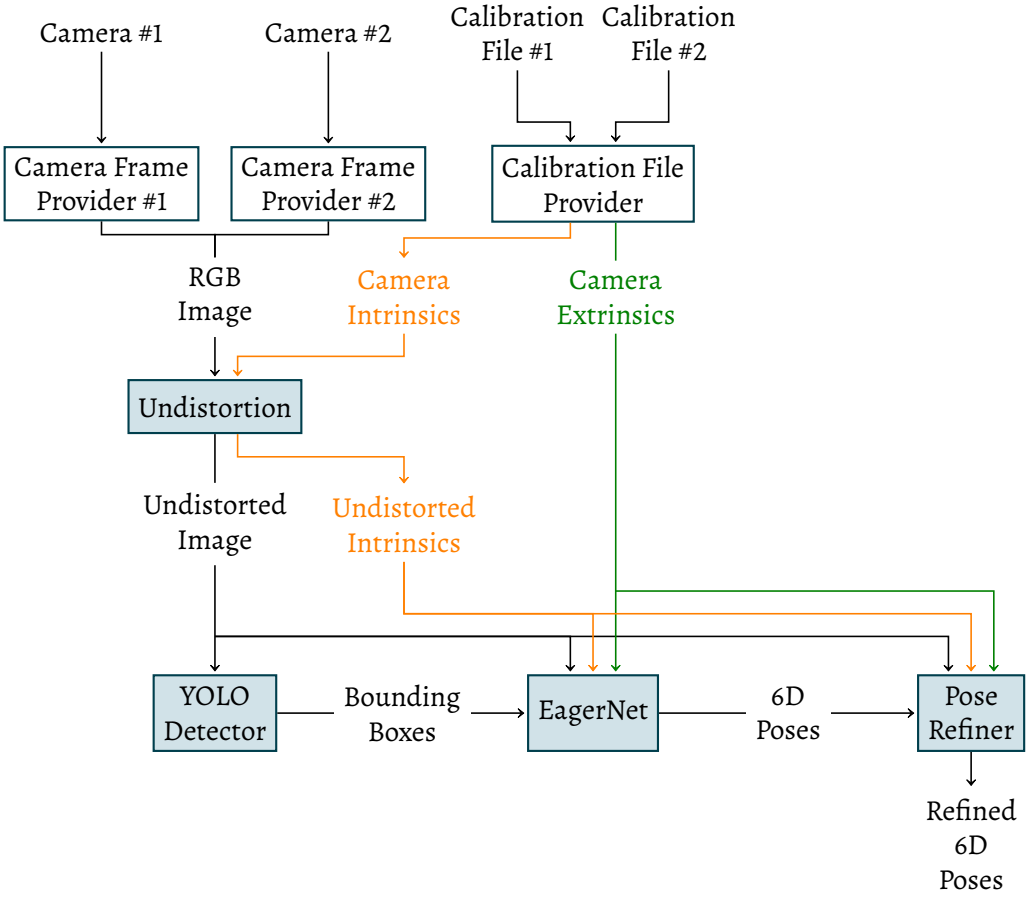


Figure 5.1: Schematic depiction of the implemented pose estimation pipeline. Given a camera image and the associated calibration file, the pose estimator returns one or more 6D pose estimates, depending on the number of target objects present in the scene. The pipeline consists of four core modules: the Undistortion module, a YOLO object detector, a pose estimator based on EagerNet [21], and the Pose Refiner for increased accuracy.

Camera Frame Provider and Calibration File Provider

The pose estimation pipeline was developed in a modular way to enable easy switching between different cameras. Two Camera Frame Provider modules were created that read the latest camera frame from either the Intel RealSense or the Microsoft Azure Kinect camera using their respective Python libraries. The frame is converted to a PCVP-compatible format and published to the pipeline. Only one of the Frame Provider modules has to be started at runtime, depending on which camera the user wants to use.

In addition, a Calibration File Provider module was developed. It uses the intrinsic and extrinsic camera parameters from the *.cal* file that was obtained through the calibration process in Chapter 4.4 and formats them into the PCVP message format. For this, the extrinsic rotation matrix and translation vector have to be merged into a transformation matrix, and the camera matrix and image resolution must be represented in a single vector. Depending on which Camera Frame Provider the user wants to use, the Calibration File Provider has to be configured to load the correct *.cal* file.

Undistortion Module

Something that needs to be considered when using physical cameras is that real lenses usually introduce distortion to images [32] [34] [35]. However, EagerNet and the Pose Refiner do not take any distortion effects into account. An established way to deal with such a problem in computer vision algorithms is to undistort the camera image by remapping each pixel from the distorted image [34] [35] [33]. The mapping function is derived from a distortion model, which is described by distortion coefficients that need to be determined through the camera calibration process. For example, the distortion model that is used in OpenCV is described in [34]. In addition to radial and tangential distortion, it can also take perspective distortion due to image sensor tilt into account.

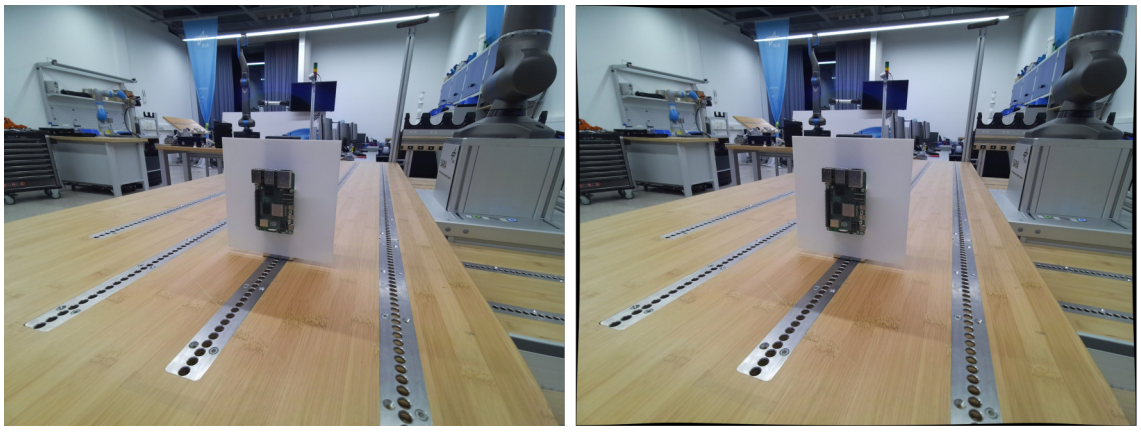


Figure 5.2: A distorted image captured using the Microsoft Azure Kinect camera (left) and the same image after undistortion (right). The effects of the undistortion are especially visible close to the edges of the image on the right, where it introduces black visual artifacts.

Following the steps described in [35], the Undistortion module was implemented for the pose estimation pipeline. Apart from the latest camera frame, the module takes the camera intrinsics as an input, which consist of the camera matrix and the distortion coefficients. In a two-step process, this data is used to calculate a new camera matrix for the undistorted image and to then transform the original image to compensate for lens distortion. The new image and the new camera matrix are provided as PCVP messages. Because the new image is considered to be undistorted, the distortion coefficients can be removed from the camera intrinsics. An exemplary result can be seen in Figure 5.2.

YOLO Detector

In the undistorted image, the target object now has to be located. To achieve this, a YOLOv7 object detector [43] was integrated into the pipeline. This particular version of YOLO was chosen because it was already available as a PCVP module, simplifying implementation. The module takes the latest undistorted camera frame as an input and detects all instances of the target object in the frame. For each detection, a bounding box around the object is defined by the position and size of the rectangle in image space. The module returns a list of bounding boxes, which can also be empty if no object was detected.

The YOLO model was trained on synthetic data, which was generated using BlenderProc2 [44]. BlenderProc2 is a photorealistic rendering software that is based on the graphics software Blender[10]. Its main purpose is to generate realistic images for the training of neural networks. The software and documentation are publicly available [45].

To generate YOLO training data, BlenderProc2 requires a 3D model of the target object. The object is placed into a random virtual scene, together with a set of unrelated distraction objects to simulate clutter in the scene. The scenes and clutter objects are included in BlenderProc2 and do not have to be provided by the user. A virtual camera now records the scene from multiple random angles and distances. The rendered images are saved together with the bounding box annotations, which are stored in a text file using YOLO compatible formatting. Apart from the number of scenes and images, the user has to specify the intrinsic parameters and the image resolution for the virtual camera.

For training the YOLO model in this work, the Raspberry Pi 3D model from [46] was used to obtain realistic training images. It is colored and provides accurate textures, in contrast to the model that is provided by the manufacturer [40]. BlenderProc2 was configured to render 1000 scenes with 25 images each, resulting in 25000 images in total. In order to generate images with similar properties as the physical camera images used for inference, the virtual camera for BlenderProc2 was configured using a scaled camera matrix from the rectified intrinsics of the Microsoft Azure Kinect camera. This modification was necessary because of the high memory requirements when training a YOLO model. Even with access to powerful hardware, training with images at a resolution of 4096x3072 pixels is not feasible. Generating 25000 images with such a high resolution would also require a lot of storage space[11]. Thus, the virtual BlenderProc2 camera was configured to record images at a lower resolution of 1024x768 pixels, the original

[10] `https://www.blender.org/` (visited on 04/30/2026)

[11] Rendering one scene with this resolution showed that a single training image takes up between two and five MB. For 25000 images, this would result in a training dataset size of 50-125 GB.

resolution was scaled by a factor of 0.25 in x- and y-direction. The camera matrix has to be scaled accordingly, which was done by multiplying the focal lengths and principal point parameters with the same factor as described in [34]. An example of the generated training images can be seen in Figure 5.3. The YOLO model was trained for 200 epochs on a computer equipped with an Nvidia RTX 3090 graphics card.

The trained model performed reliably, even when using the Intel RealSense camera instead of the Microsoft Azure Kinect camera on the real system. For this reason, it was not necessary to retrain the model with rendered images that were generated using the Intel RealSense intrinsics.

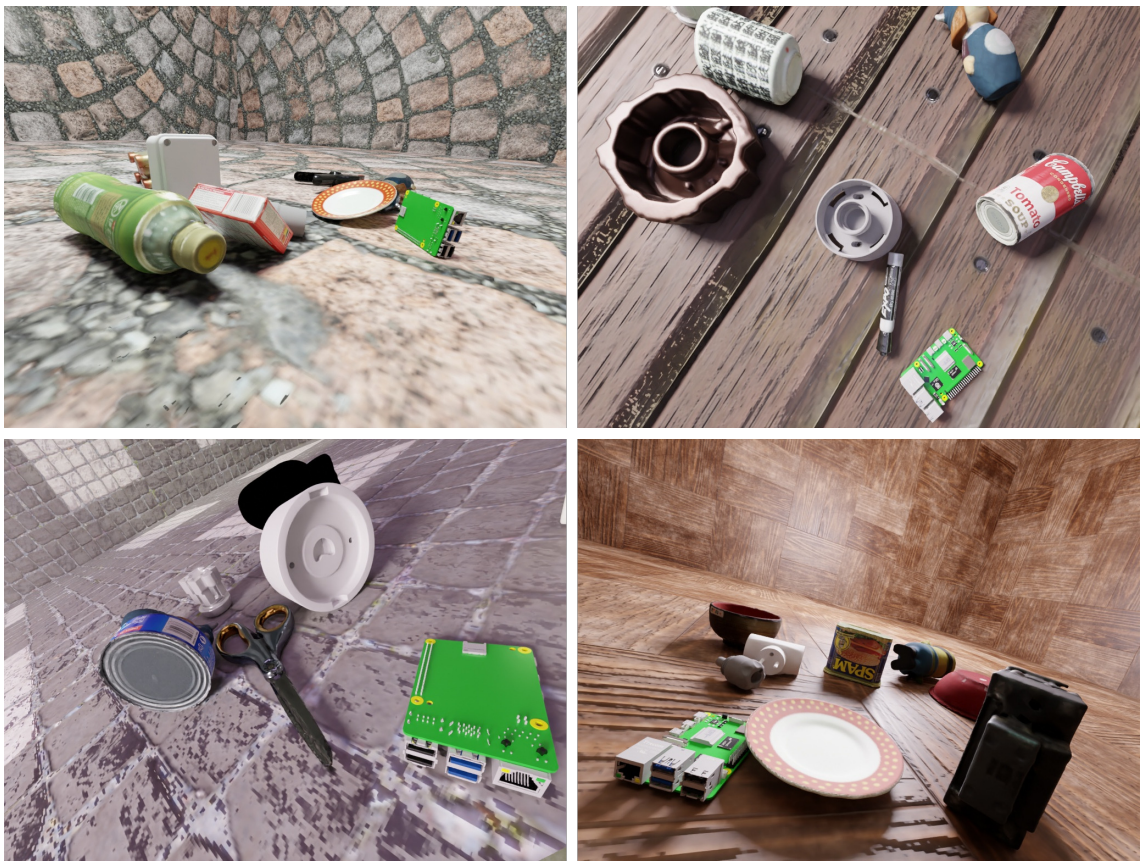


Figure 5.3: Examples of the YOLO training images generated by BlenderProc. Using a virtual camera with similar properties as the Microsoft Azure Kinect, the target object is rendered in different scenes and from multiple angles. Various unrelated 3D models are also included in the scene to simulate clutter.

EagerNet Module

After the target object was detected in the image, the pose can be estimated using EagerNet [21]. The algorithm was already available as a PCVP module and only had to be included in the pipeline. It requires a list of bounding boxes from the YOLO Detector as input, as well as the camera image on which the detection was performed, and the associated camera

matrix and extrinsic camera parameters. In the previous step, the detection was done on the undistorted camera image, which is why this image is provided to the EagerNet module, together with the camera matrix that was returned by the Undistortion module. As described in [21], this image is cropped to the content of a bounding box and used as the input for EagerNet, which returns a pose estimate. This process is repeated for every bounding box in the list. The module then outputs a list of all pose estimates (one for every input bounding box) as a PCVP message.

At the time of writing, EagerNet had not been made accessible to the public yet. For training EagerNet, an internal training toolkit was available that uses BlenderProc2 to generate annotated rendered images of the target object in different poses. This process is similar to how the YOLO training data was generated, but the images are annotated with object poses instead of bounding boxes. The user only needs to provide a 3D model of the target object. Again, the Raspberry Pi model from [46] was used.

However, before training, the center of the model was moved to align with the Ethernet port. This was done by importing the model into the 3D software Blender, transforming its position, and exporting it again. The result is shown in Figure 5.4. This way, the pose estimates of a trained EagerNet model represent the Ethernet port poses, without the need for an additional transformation. By default, the 3D model center is located at the center point of the bottom circuit board surface (see Figure 5.4). The exact transformation to the center of the Ethernet port opening can be determined using the mechanical drawing of the device [40].

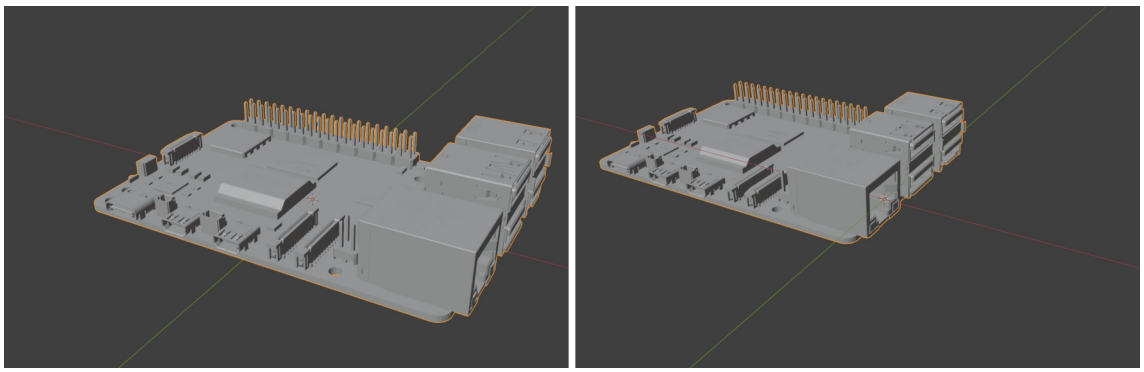


Figure 5.4: The Raspberry Pi 3D model imported into the 3D software Blender. The default model center is shown in the left image. In the right image, the model was transformed to have its center at the Ethernet port opening.

Pose Refiner

To improve the pose estimates returned by EagerNet, the idea from [21] is adopted to use a region-based method that aims to distinguish between the target object and the image background. As in [21], the Region-Based Gaussian Tracker (RBGT) presented in [47] and [22] is integrated. Although RBGT was originally designed to track moving targets in a series of camera frames, its iterative nature and local optimization strategy result in increasingly accurate pose estimation when applied repeatedly to the same camera frame.

This property was also exploited in [47], where the pose estimators *PoseCNN* [48], *AAE* [10], and *CosyPose* [49] were examined. Using RBGT as a refiner improved the pose estimation for all three algorithms.

RBGT is available for public use [50]. To integrate RBGT into the pose estimation pipeline, the Pose Refiner module was developed. It implements a Python wrapper that simplifies the usage of RBGT as a refiner.

Since the refiner operates on the undistorted camera frame as well, the undistorted image, the undistorted camera matrix, and the camera extrinsics are used as input for the module. Additionally, the pose estimates from the previous step are required as initial estimates for RBGT. The tracker works by using image statistics to distinguish the target object from the background and finds a pose that explains this segmentation based on rendered images of the target [47] [22]. Therefore, it is not necessary to train the refiner, the user only needs to provide a 3D model of the target object. Again, the transformed Raspberry Pi model from [40] was used (see Figure 5.4). Finally, the refined poses are published using LN. Considering that the camera was calibrated in robot base frame, these estimates are given in robot base frame as well.

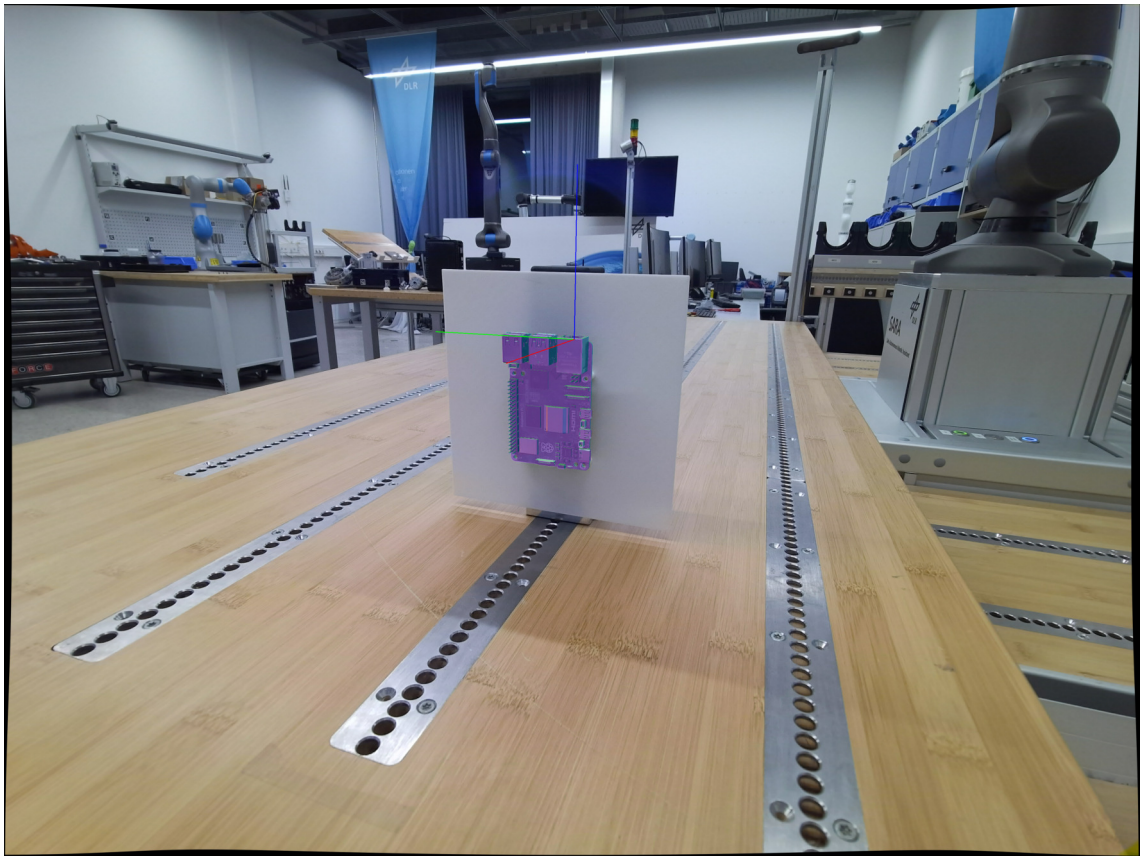


Figure 5.5: The final pose estimate after refinement. The pose estimation was done on the camera frame shown in Figure 5.2. To provide a qualitative indication of pose estimation accuracy, a 3D model of the Raspberry Pi is rendered on top of the camera frame.

Optionally, it is possible to display a rendered model of the target object on top of the camera frame, a feature that is provided by the RBGT Python wrapper. This allows the user to intuitively evaluate the precision of a pose estimate. An example is given in Figure 5.5.

It should be noted that RBGT can be configured to include depth images, which the Intel RealSense and Microsoft Azure Kinect cameras can both provide. The pose estimation pipeline was extended with this feature, but there was no measurable improvement in pose estimation accuracy. To simplify the pipeline, it was reverted to use only 2D images.

Covariance Generation

The pose estimation covariance that is required to include uncertainty-aware pose estimates into EU-KGRL, is determined empirically. The idea is to model the pose estimates as a multivariate normal distribution. It is inspired by the fact that RBGT uses a likelihood function to optimize its predictions, which follows a Gaussian distribution [22] [47].

In fact, the RBGT Python wrapper does provide a covariance matrix for every pose estimate, which models the uncertainty of the prediction. However, for the pose estimation pipeline in this thesis, the uncertainty covariance matrix from RBGT was not used. The reason for this is that it only captures the pose estimate uncertainty in the current camera frame, ignoring factors like image noise. Initial experiments with RBGT that were done for this thesis, showed that the covariance generated by RBGT stays consistent when repeating the estimation process on the same camera frame. However, when repeating the estimation process on a new camera frame under identical conditions (same viewing angles and lighting conditions), the covariance matrix output differs.

Instead of relying on a single camera frame, a separate Python script was developed that captures 50 frames of the target object under the same conditions (same viewing angles and lighting conditions). For each of these frames, the implemented pose estimation pipeline performs a prediction. From the positions of these poses, a covariance matrix $\Sigma_g \in \mathbb{R}^{3x3}$ is constructed using the Python library *NumPy*[12]. As mentioned in Chapter 1.2, only the inclusion of positional variance is examined in this work. However, the construction of an orientation covariance matrix could be done in a similar way, for example, by using an Euler-angle representation of the target object orientation[13].

The covariance matrix Σ_g is constructed before EU-KGRL is used, it is assumed that the target pose does not change afterward. In a more dynamic real-world scenario, a new covariance could be obtained with every new pose estimate. Modern cameras can provide up to 30 frames per second [37] and higher [38], and the implemented pose estimation pipeline only takes a few milliseconds to generate a prediction.

Using an LN service, any Python script can trigger the generation of a new pose estimate. The service responds with the latest estimate and the corresponding covariance matrix Σ_g.

[12] https://numpy.org/doc/2.2/reference/generated/numpy.cov.html (visited on 04/30/2026)

[13] In this case, the problem of angle wrapping must be considered, so that a jump from 360° to 0° doesn't lead to a high variance.

5.2 Extension of the Simulation Environment

As a proof of concept, EU-KGRL was evaluated in a simplified 2D simulation environment. For comparability, the Python simulator introduced in [8] was reused, which was specifically designed to test and analyze KGRL. In the original formulation, the simulation consists of a 2D environment in which a robot must reach a predefined goal position while traversing a constrained workspace containing a narrow passage [8]. It is presented in Figure 5.6.

Nine demonstrations $D = \{\{t_{n,m}, p_{n,m}\}_{n=1}^{400}\}_{m=1}^{9}$ are provided that navigate the robot through the passage to reach the goal (see Figure 5.6). Each demonstration has a length of 400 time steps and maps every time step t to a robot position p. After recording the demonstrations, an obstacle and a restricted zone are added to the simulation that the robot has to avoid. The object is placed in a way that an unmodulated KMP trajectory intersects with it. Moving against one of the walls that form the passage resets the robot to its previous position, simulating physical resistance. The task is completed successfully when the goal is reached within 400 time steps without being stuck at the walls for 20 or more time steps. The goal is considered to be reached when the distance between goal position and robot position is smaller than 0.03 units.

For this thesis, the Python simulation was extended to introduce external variance into the system that does not depend on the demonstrations. In the original simulation, the goal pose is fixed and known. This assumption is relaxed to better reflect the intended real-world application, where the target pose is not known exactly but determined by a pose estimator. To simulate this, the ground truth target pose is no longer static. Instead, it is sampled at the beginning of each episode from a multivariate Gaussian distribution. Let $\mu_g \in \mathbb{R}^2$ denote the estimated target pose provided by a hypothetical vision system, and let $\Sigma_g \in \mathbb{R}^{2\times2}$ denote the associated covariance matrix representing pose uncertainty. The true target pose g is defined as:

$$g \sim \mathcal{N}(\mu_g, \Sigma_g),$$

where μ_g corresponds to the original, fixed goal pose used in the baseline simulation, and Σ_g is manually specified to encode external uncertainty. In code, this was done using numpy's *multivariate_normal()* function[14]. The parameters were set to the following specific values:

$$\mu_g = \begin{pmatrix} 0.469 \\ 0.088 \end{pmatrix}, \quad \Sigma_g = \begin{pmatrix} 0.001 & 0 \\ 0 & 0.00025 \end{pmatrix}.$$

The value for μ_g was taken from the source code of the original simulation [8]. Σ_g was chosen to have a high variance in the x-direction and a lower variance in the y-direction. This modification introduces variability in the target pose and allows it to deviate far from the demonstrations, in particular due to the high variance in the x-direction. Consequently, the learned movement primitive does not account for such uncertainty, requiring the agent to adapt its behavior accordingly.

[14] https://numpy.org/devdocs/reference/random/generated/numpy.random.multivariate_normal.html (visited on 04/30/2026)

To provide an intuitive visualization of the introduced uncertainty, the target pose distribution is illustrated in the simulation using covariance ellipses. In particular, ellipses corresponding to one and two standard deviations (1σ and 2σ) are plotted around the mean μ_g. The modified simulation with the visualized target pose region is shown in Figure 5.6.

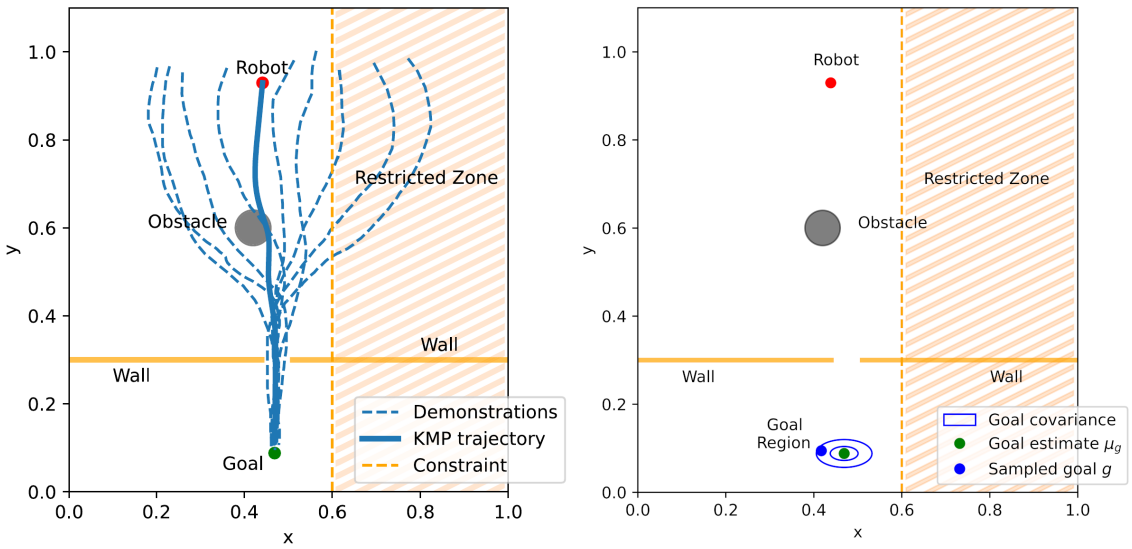


Figure 5.6: Left: the original simulation with nine recorded demonstrations, the image was taken from [8]. Right: the extended simulation developed for this thesis, in which the goal g is sampled from a distribution.

The task definition is updated to reflect this change. An episode is considered successful only if the robot reaches a state sufficiently close to the actual sampled target pose g, rather than the nominal estimate μ_g. Episodes terminate unsuccessfully if the robot becomes blocked in the narrow passage for 20 or more consecutive time steps, or if a maximum horizon of 450 time steps is reached without achieving the goal. The horizon is increased, because the sampled target position may be farther away from the initial robot position as the original goal μ_g, which was observed during demonstration.

In addition, the goal-reaching condition is refined to better reflect continuous robot motion. In the original simulation, success is determined by checking whether the Euclidean distance between the robot and the goal falls below the predefined threshold of 0.03 units at each discrete time step. The formulation implicitly assumes point-wise state updates and can penalize larger motion increments that would cause the robot to overshoot the goal. To address this limitation, the goal condition is reformulated by considering the robot's motion between consecutive time steps. Instead of evaluating only the current position, the minimum distance between the target pose g and the line segment connecting the previous and current robot positions is computed and compared to the threshold. The approach effectively models the robot as moving continuously along this segment rather than jumping between discrete states. As a result, target-reaching events are correctly detected even when the robot performs larger and faster movements in the vicinity of the goal.

5.3 External-Uncertainty KGRL (EU-KGRL)

As described in Chapter 3.2, KGRL uses an LC-NS-KMP to generate a nominal policy learned from demonstrations, which is modified by a reactive RL policy in accordance with the variance in the demonstrations [8]. This variance is provided by a Gaussian Mixture Model, which is constructed from the demonstrations. Using Gaussian Mixture Regression, a probabilistic reference trajectory $T_r = \{\hat{\mu}_n, \hat{\Sigma}_n\}_{n=1}^{N}$ is calculated, where N is the trajectory length and $\hat{\mu}_n$ and $\hat{\Sigma}_n$ are means and covariance matrices for each time step. $\hat{\mu}_n$ and $\hat{\Sigma}_n$ are used by LC-NS-KMP to generate the nominal trajectory, and $\hat{\Sigma}_n$ is embedded into the soft null-space projector to scale the intensity of the reactive nominal policy offset for every time step.

The proposed extension of KGRL decouples uncertainty estimation from demonstrations by introducing externally-provided covariance to the LC-NS-KMP formulation. This extension, termed External-Uncertainty KGRL (EU-KGRL), enables operation in environments where demonstrations are not representative of execution uncertainty. For now, it is assumed that the external variance does not change over time. To do this, the reference trajectory T_r obtained from the Gaussian Mixture Model is replaced with a modified reference trajectory T_r' where:

$$T_r' = \{\hat{\mu}_n, \Sigma_n'\}_{n=1}^{N} \tag{5.1}$$

$$\Sigma_n' = \begin{cases} \hat{\Sigma}_n, & \text{if } n \leq t_s \\ C_{mod} \cdot \Sigma_E, & \text{otherwise} \end{cases} \tag{5.2}$$

Here, $\hat{\mu}_n$ and $\hat{\Sigma}_n$ are the original mean and covariance from Gaussian Mixture Regression, and Σ_E is the desired external covariance. t_s is the time step at which the demonstration covariance is replaced with the external covariance and C_{mod} is a factor by which the covariance matrix can be scaled component-wise. After replacing the reference trajectory T_r with the modified reference trajectory T_r', the LC-NS-KMP output is computed as described in [8] and Chapter 3.1, without any changes. Therefore, for this thesis, the original software used in [8] to train KGRL was easily extended with this adaptation.

The inclusion of t_s allows EU-KGRL to switch between the two covariances and only consider the external covariance after time step n. For a connector-insertion task with pose estimation uncertainty, this switch would happen when the robot is close to the connector port. Following the principle of guided reinforcement learning, the idea is to only include the external variance when it is relevant for the robot, and default to the demonstration variance otherwise. This becomes important when the robot has to follow a longer trajectory before reaching its goal, and might even has to follow secondary objectives like obstacle avoidance. The extended simulation setup that was presented above (Section 5.2) considers such a scenario (also see Figure 5.7). Choosing $t_s = N$ would result in the original KGRL formulation, while $t_s = 0$ would lead EU-KGRL to consider the external variance right from the beginning. Depending on the use case, the formulation of Σ_n' could also be changed to introduce a time interval in which the external covariance is used. As Σ_n' can be modified in every time step, the inclusion of a time-dependent covariance $\Sigma_E(n)$ is possible as well. For the use cases examined in this work, the formulation of Σ_n' given in Equation 5.2 is sufficient.

In KGRL, the covariance $\hat{\Sigma}_n$ obtained from the reference trajectory T_r shapes the exploration [8]. In regions with high variance in the demonstrations, the output of the reactive RL policy is highly impactful. In regions with little demonstration variance, the RL policy is barely able to modulate the nominal trajectory. Although the exploration space and the area covered by the demonstrations are related, they are not necessarily identical. An example of this effect is demonstrated in Figure 5.7 where the exploration space is visibly smaller than the area covered by the demonstrations. The example shows exploration trajectories in the original KGRL simulator and was taken from [8]. For regular KMP, this is usually not a problem, as the goal is to learn small corrections for the nominal trajectory. The covariance $\hat{\Sigma}_n$ enforces the robot to closely follow the nominal trajectory for certain time steps n, in which there is little variance in the demonstrations. It is not tied to any physical property of the environment.

However, this might be a problem when introducing external covariance, where it could be necessary to make bigger corrections to the nominal trajectory. For this reason, the covariance modifier C_{mod} is introduced in Equation 5.2. The parameter allows to scale the covariance-shaped exploration for the time steps in which the external covariance is considered. It needs to be fine-tuned by the user depending on the use case. For example, if the external covariance matrix represents target pose uncertainty, the user can set a higher value for C_{mod} to ensure that the target can always be reached by the robot.

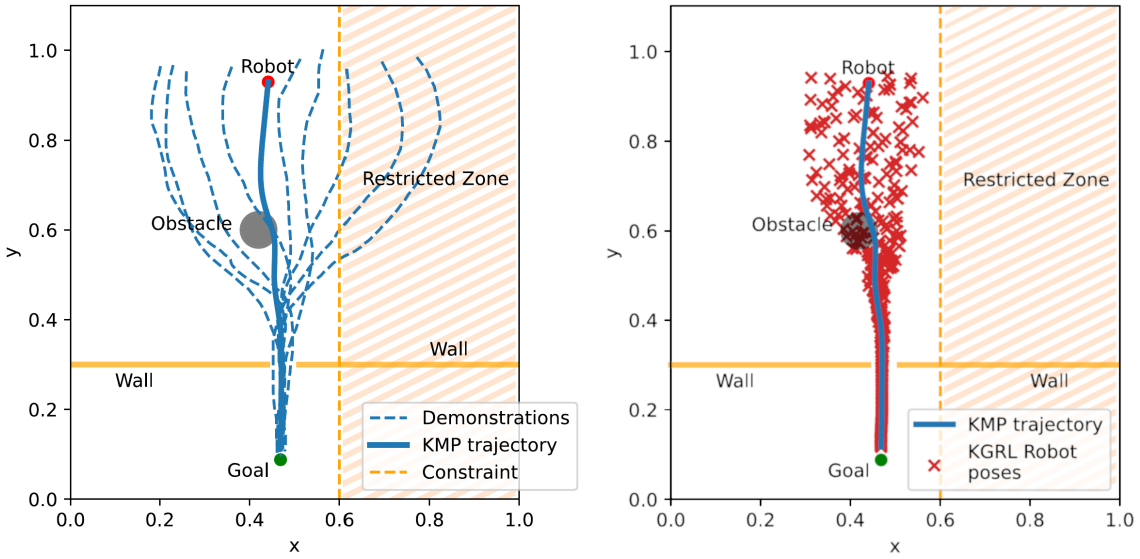


Figure 5.7: Simulated 2D environment from [8]. Based on nine demonstrations, LC-NS-KMP generates a nominal KMP trajectory (left). KGRL learns to modulate this trajectory based on variance in the demonstrations (right). However, the area explored by the robot is smaller than the region covered by the demonstrations. This is particularly visible at the start of the trajectory. Both plots were taken from [8].

Using the definition of Σ_n' from Equation 5.2, EU-KGRL completely disregards the covariance obtained from demonstrations when $n > t_s$. This design choice is deliberate and motivated by the characteristics of the use case that was examined in this thesis. As will be shown in Chapter 6.3 (see Figure 6.14), the variance from demonstrations for a

connector-insertion task is very low at the end of a reference trajectory. A similar effect can be seen in the simulation example in Figure 5.7. During training, the robot barely deviates from the nominal trajectory when approaching the goal. While it was tested to add the scaled external covariance to $\hat{\Sigma}_n$ for $n > t_s$ instead of replacing it, the effect was negligible. For simplicity, the simpler formulation in Equation 5.2 was chosen.

Using the modified reference trajectory T_r', $\hat{\Sigma}_n$ is replaced with $\Sigma_n^{'}$ and the *constrained mean minimization subproblem* for LC-NS-KMP (see Chapter 3.1) becomes[15]:

$$\begin{aligned} \underset{\mu_w}{\operatorname{argmin}} \sum_{n=1}^{N} \frac{1}{2} \left(\Theta^\top(s_n)\mu_w - \hat{\mu}_n\right)^\top {\Sigma_n^{'}}^{-1} \left(\Theta^\top(s_n)\mu_w - \hat{\mu}_n\right) \\ + \frac{1}{2}\lambda\, \mu_w^\top \mu_w + \frac{1}{2}\beta(\mu_w - \hat{\mu}_w)^\top(\mu_w - \hat{\mu}_w), \\ \text{s.t. } g_{n,f}^\top \eta(s_n) \geq c_{n,f}, \forall f \in \{1,2,\dots,F\}, \forall n \in \{1,2,\dots,N\}. \end{aligned} \tag{5.3}$$

By solving the optimization problem as described in [8] and in Chapter 3.1, the LC-NS-KMP formulation can be computed as:

$$\mathbb{E}(\eta(s)) = k^* A^{'} \mu + k^* A^{'} \Sigma^{'} \bar{G} \alpha + \frac{\beta}{\gamma}(\hat{k}^* - k^* A^{'} \hat{K}) \underline{K}^{-1} \xi, \tag{5.4}$$

where

$$A^{'} = (K + \lambda \Sigma^{'})^{-1},$$

with the notable change that $\Sigma^{'} = \text{blockdiag}(\Sigma_1^{'}, \Sigma_2^{'}, \dots, \Sigma_N^{'})$. With these adjustments in mind, the generation of an EU-KGRL trajectory is adopted from the KGRL formulation in [8] and defined as:

$$\mathbb{E}(\eta(s)) = k^* A^{'} \mu + k^* A^{'} \Sigma^{'} \bar{G} \alpha + \frac{\beta}{\gamma}(\hat{k}^* - k^* A^{'} \hat{K}) \pi(\xi|q). \tag{5.5}$$

Based on the formulation in [8], EU-KGRL is summarized in Algorithm 1. As in KGRL, the user gives multiple demonstrations and defines all KMP-specific hyperparameters, the set of constraints, and the RL policy parameters. In addition, the newly introduced parameters t_s and C_{mod} have to be set. Before training starts, the pose estimator (or the simulation) provides EU-KGRL with the external covariance Σ_E. In every training episode, a new target estimate $\tilde{g}$ (modeled by Σ_E) is being collected from the pose estimator (or simulation). Note that this step is only necessary for the proposed use case in this thesis, in a different setting, the uncertainty could be an inherent property of the environment or the system. For every time step, the reference trajectory T_r has to be modified according to 5.2, in order to obtain $T_r^{'}$. Using $T_r^{'}$ and a null-space action ξ generated by the RL policy π, the EU-KGRL trajectory $\mathbb{E}(\eta(s))$ is calculated with 5.5. After applying the trajectory to the robot, the reward is computed, and the RL policy is updated.

[15] Here, $g_{n,f}$ represents a linear constraint vector following the notation in [8], not to be confused with the notation for the target pose that is used throughout this thesis.

Algorithm 1 External-Uncertainty Kernelized Guided RL (EU-KGRL).

Collect demonstrations $D \leftarrow \{\{s_{n,m}, \eta_{n,m}\}_{n=1}^{N}\}_{m=1}^{M}$
Set hyperparameters λ, β and define kernel function $k(.,.)$
Set horizon h
Model joint probability distribution $P(s,\eta)$ using GMM
Define set of constraints $O = \{\{g_{n,f}^{\top}, c_{n,f}\}_{n=1}^{N}\}_{f=1}^{F}$
Initialize a RL policy $\pi(\xi|q)$ and policy parameters
Set t_s and C_{mod}
Collect external covariance Σ_E from pose estimator or simulation
loop for each episode
    Collect target estimate $\tilde{g}$ from pose estimator or simulation (optional)
    loop for each time step $n = 1, \dots, N$
        In current state s_n, retrieve the reference trajectory distribution $T_r \leftarrow \{\hat{\mu}_i, \hat{\Sigma}_i\}_{i=n}^{n+h}$ from GMR
        Compute $\Sigma_n^{'}$ with 5.2
        Compute the modified reference trajectory distribution $T_r^{'}$ with 5.2
        Choose constraints for $T_r^{'}$ from O
        Sample RL action ξ from RL policy $\pi(\xi|q)$
        Compute Lagrange multiplier vector α as in [8]
        Compute $\mathbb{E}(\eta(s_n))$ with 5.5
        Apply $\mathbb{E}(\eta(s_n))$ to robot
        Compute reward
        Update RL policy π
    end loop
end loop

6

Experiments and Results

This chapter covers simulation and real-robot experiments, including their results. Prior to robot testing, the pose estimation pipeline's position accuracy was analyzed.

6.1 Experiments in Simulation

For the experiments in simulation, KGRL and EU-KGRL are configured with the RL policy $\pi(\xi|p_t^\mu)$ that generates null-space actions $\xi \in \mathbb{R}^2$ based on the current robot position p_t^μ in the estimated target frame:

$$p_t^\mu = p_t - \mu_g, \quad \mu_g = \begin{pmatrix} 0.469 \\ 0.088 \end{pmatrix},$$

where $p_t \in \mathbb{R}^2$ is the robot position in world frame at time step t.

The RL policy $\pi(\xi|p_t^\mu)$ was configured as a Deep Neural Network (DNN) consisting of 2 hidden layers with 256 neurons each. The training process was implemented using the Python library Stable-Baselines3 [51], where the algorithm Truncated Quantile Critics (TQC) was chosen [52] [53]. It was configured with the following parameters:

```
learning rate = 0.001,
soft update coefficient = 0.02,
discount factor = 0.99,
training frequency = 8,
gradient steps = 8,
entropy regularization coefficient = auto
```

The reward function r_t is defined as follows:

$$r_t = r_a + r_o + r_T, \tag{6.1}$$

$$r_a = -0.1 \delta p_t^\top \delta p_t, \tag{6.2}$$

$$r_o = \begin{cases} -100(0.04 - d_t) & \text{if } d_t \leq 0.04 \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

$$r_T = \begin{cases} 200 & \text{at terminal step } T\text{, if successful} \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

where r_a is the action cost, r_o is the obstacle avoidance cost and r_T is the terminal reward. δp_t is the displacement of the robot from its previous position ($\delta p_t = p_t - p_{t-1}$) and d_t is the distance of the robot to the obstacle.

It should be noted that this configuration is identical to the one from the original simulation in [8], with two changes. First, the input for the policy is p_t^μ instead of t. This adjustment was motivated by the fact that the robot has to learn to explore certain areas of the simulation, for which its position is a more useful policy input than the current time step. Additionally, the robot position was also provided in the real-world experiments presented in the original KGRL paper [8]. As p_t^μ represents the robot position relative to the virtual target estimate μ_g instead of the true target position, this modification also does not provide the robot with any information it would not have in a real-world scenario. Therefore, this change does not make the simulation setup any less comparable.

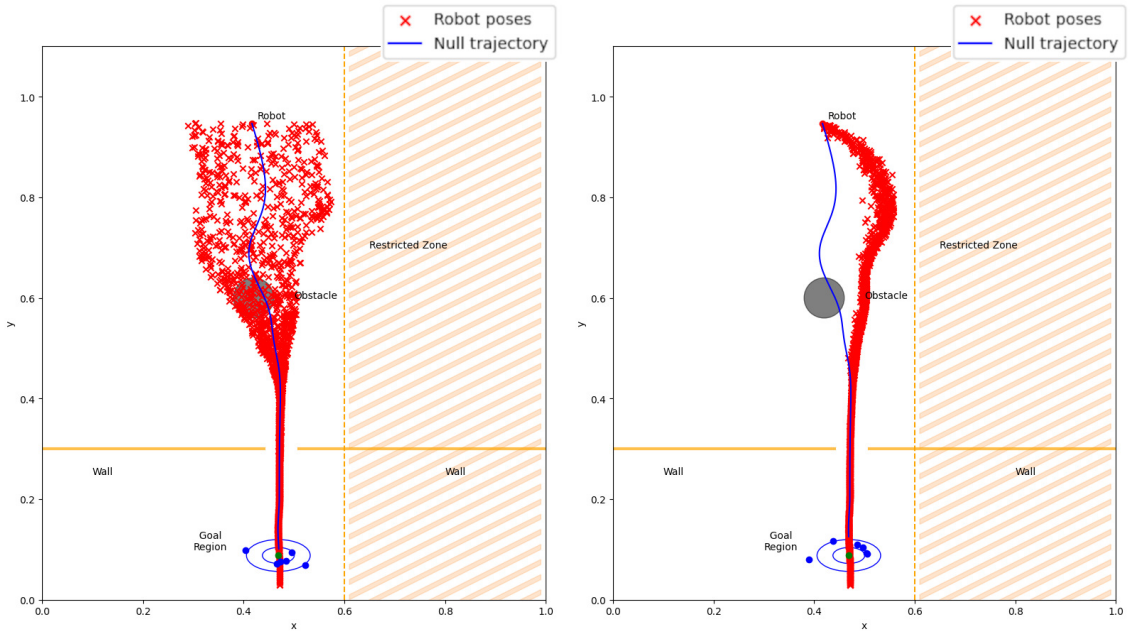
The second change that was made is the adjustment of the action cost modifier in Equation 6.2, which was reduced from 10 to 0.1. The reason is, that in the extended simulation, the robot now has to make bigger movements in order to solve the task, especially in the target region. In first experiments, the RL policy failed to learn the task because the action cost outweighed the terminal reward. Reducing the action cost modifier to 0.1 punishes exploration less, encouraging the robot to move more if it helps to solve the task.

The LC-NS-KMP generates a new robot position p_t for every time step t, while considering the null-space action ξ from the RL policy $\pi(\xi|p_t^\mu)$. The LC-NS-KMP is constructed from the same nine demonstrations that were used in [8] (also shown in Figure 5.6). The linear constraints were adopted from [8] as well. A new random target goal g is sampled for every episode. To compare KGRL and EU-KGRL, both RL algorithms were trained ten times, with 250 episodes for each run.

KGRL

As a baseline algorithm, KGRL was trained on the modified simulation environment with the setup described above. Figure 6.1 shows the exploration behavior of a KGRL agent before and after training for 250 episodes. In both cases, the robot's poses over five episodes are shown. Additionally, the null trajectory is presented, which is the unmodulated LC-KMP trajectory that would be generated by KGRL if the null-space action ξ were zero for every time step.

Figure 6.1b shows that KGRL learns to avoid the obstacle, while not entering the restricted zone. However, close to the goal region, it does not modulate the null trajectory strongly enough to reach the sampled target, which is shown in blue. This observation is supported by the results in Figure 6.3, which shows the average overall reward (Equation 6.1), average action cost (Equation 6.2), average obstacle avoidance cost (Equation 6.3), and average success rate for KGRL over 250 episodes. A moving average window over ten episodes was used for the presentation of the training performance. While the obstacle avoidance cost and the action cost quickly converge, the overall reward does not improve over time, and the success rate stays below 0.7 for most episodes. Given the restricted exploration space due to little variance in the demonstrations, KGRL does not learn to solve the task reliably.



(a) KGRL: Episodes 0-5

(b) KGRL: Episodes 250-255

Figure 6.1: Exploration behavior of the robot using KGRL over five episodes, before training (a) and after training (b) for 250 episodes. KGRL manages to learn obstacle avoidance, but not to reach the goal reliably. The blue dots show the sampled goal positions g, the green dot denotes μ_g.

EU-KGRL

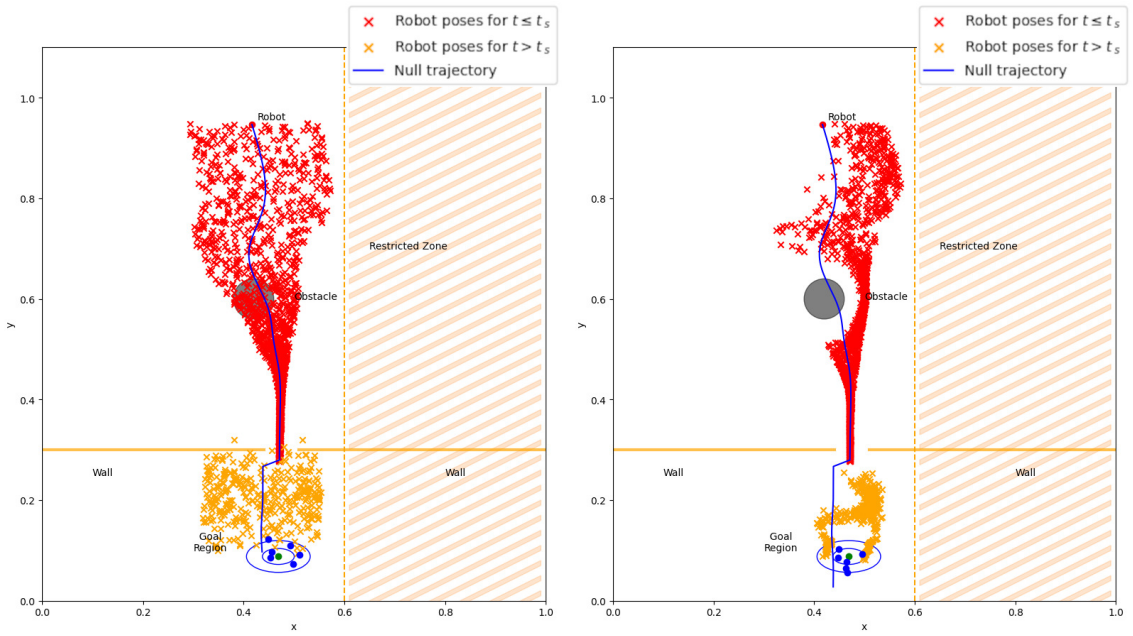
Following the formulation in Chapter 5.3, EU-KGRL was implemented into the simulation using the following parameters:

$$t_s = 284,$$
$$\Sigma_E = \Sigma_g = \begin{pmatrix} 0.001 & 0 \\ 0 & 0.00025 \end{pmatrix},$$
$$C_{mod} = 40.$$

Σ_E is set to the covariance of the target pose, which is the same covariance that is used to sample g in the simulation. t_s is set to the time step right after the robot crosses the passage when following the null trajectory, it was determined experimentally by fixing ξ to zero and logging time step t together with robot position p_t. C_{mod} is set to a relatively high value that enables the robot to reach most of the goal region, and was chosen by trial and error.

Figure 6.2 presents the exploration behavior of an EU-KGRL agent before and after training. The robot poses p_t for $t > t_s$ are shown in orange instead of red, to highlight how the exploration behavior is altered when external variance is introduced. Compared to KGRL, the exploration space of EU-KGRL is visibly bigger close to the target, covering the entire plotted goal region. In the experiment shown in Figure 6.2b, EU-KGRL learned to

solve the primary task of reaching the target by quickly moving between the left and the right sides of the goal region. However, EU-KGRL was also observed to learn alternative strategies, for example, by trying to cover the goal region uniformly. In the training run shown Figure 6.2b, EU-KGRL learned to narrow its exploration down to only cover the goal region, whereas the exploration space was visibly bigger than that region before training. Just like KGRL, EU-KGRL also learns to avoid the obstacle without entering the restricted zone.



(a) EU-KGRL: Episodes 0-5

(b) EU-KGRL: Episodes 250-255

Figure 6.2: Exploration behavior of the robot using EU-KGRL over five episodes, before training (a) and after training (b) for 250 episodes. EU-KGRL manages to learn both obstacle avoidance and to reach the goal. The blue dots show the sampled goal positions g, the green dot denotes μ_g.

The detailed plots of the reward function terms in Figure 6.3 support these findings. The mean overall reward for EU-KGRL is higher than the mean reward for KGRL throughout the entire training process. The obstacle avoidance cost graph is almost identical to the one from KGRL, showing that EU-KGRL also learns to avoid the obstacle. Considering that EU-KGRL still uses the demonstration covariance for the part of the trajectory in which the obstacle was introduced (see red exploration poses in Figure 6.2), this is not surprising. In contrast to KGRL, the success rate for EU-KGRL does converge and achieves a value of over 0.9. An interesting observation is that the success rate is already at 1 when training starts, suggesting that the initial random exploration behavior is already a good strategy to solve the primary task of reaching the target. The action cost for EU-KGRL is worse than the action cost for KGRL, as the external variance in EU-KGRL allows the robot to make bigger movements in the goal region in order to reach the target and receive a high terminal reward.

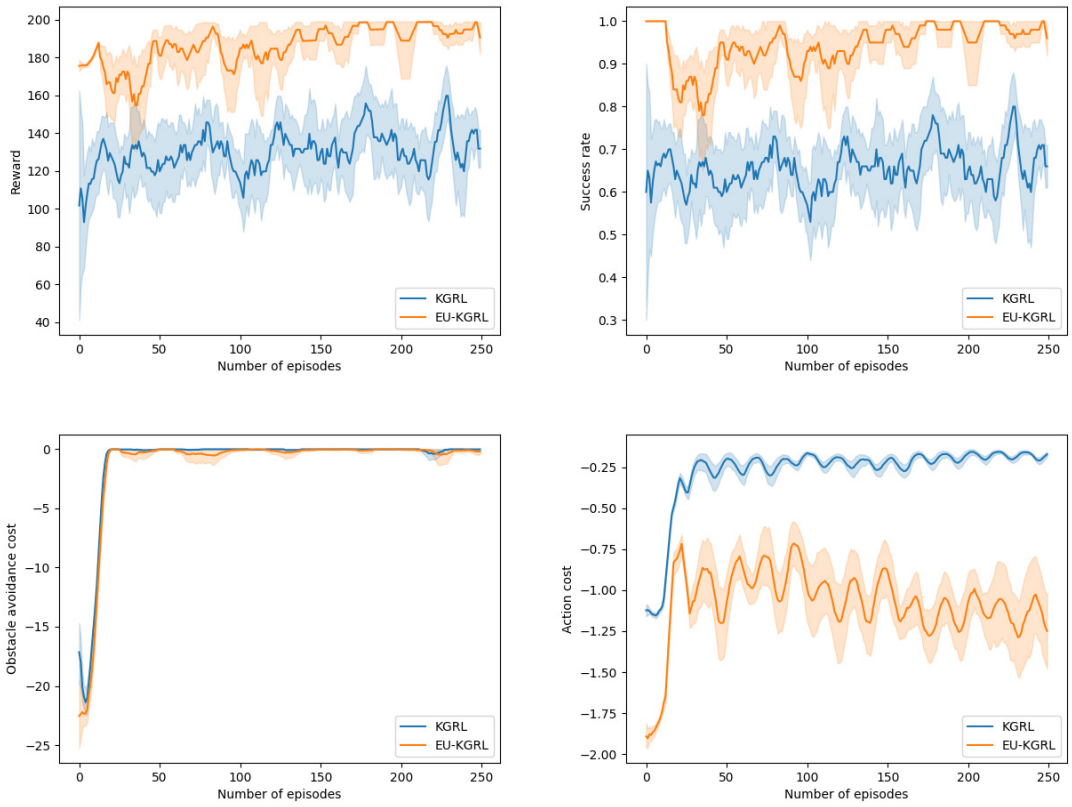


Figure 6.3: The performance of KGRL and EU-KGRL over 250 training episodes. Each algorithm was run ten times. Shown are four evaluation metrics, the overall reward (top left), success rate (top right), obstacle avoidance cost (bottom left), and action cost (bottom right), each plotted against the number of episodes. The solid lines represent the mean values, while the shaded ares indicate the 95% confidence intervals with a moving average window of size ten. KGRL and EU-KGRL both manage to learn obstacle avoidance. However, KGRL does not learn to solve the primary task of reaching the target.

EU-KGRL with Trajectory Correction

Figure 6.2 shows that the null trajectory for EU-KGRL is different to the one from KGRL demonstrated in Figure 6.1. After the robot crosses the passage (which is for $t > t_s$), the null trajectory shows an abrupt offset to the left. The null trajectory represents the output of the LC-NS-KMP without null-space modulation. Consequently, the exploration space of the robot also receives an offset to the left for $t > t_s$, as the exploration space is defined by the modulations that the RL agent can make to the nominal LC-NS-KMP trajectory.

The reason for this is that EU-KGRL introduces external variance that does not come from demonstrations. It replaces the variance of the probabilistic reference T_r that is used for the construction of the LC-NS-KMP. Not only does it influence the null-space projector, but also the KMP formulation. Increasing the variance in certain time steps decreases the KL-divergence between T_r and the kernelized trajectory model, leading to a KMP output that follows the mean of T_r less closely (see Chapter 3).

Despite the offset, EU-KGRL still solves the task. As shown in Figure 6.2b, it can learn to compensate for such an offset, if the exploration space is big enough. However, in order to examine the effect of this offset on training performance, a modified version of EU-KGRL was tested, which produces a null trajectory that follows the demonstrations more closely. This is done by splitting the LC-NS-KMP formulation into two parts: the linearly constrained KMP (LC-KMP) and the null-space projector. Considering that the EU-KGRL formulation is defined as (also see Equation 5.5):

$$\mathbb{E}(\eta(s)) = k^* A' \mu + k^* A' \Sigma' \bar{G} \alpha + \frac{\beta}{\gamma} (\hat{k}^* - k^* A' \hat{K}) \pi(\xi|q),$$

it can be rewritten as:

$$\mathbb{E}(\eta(s)) = \mathbb{E}_{LC}(\eta(s)) + \mathbb{E}_{NS}(\eta(s))$$

with

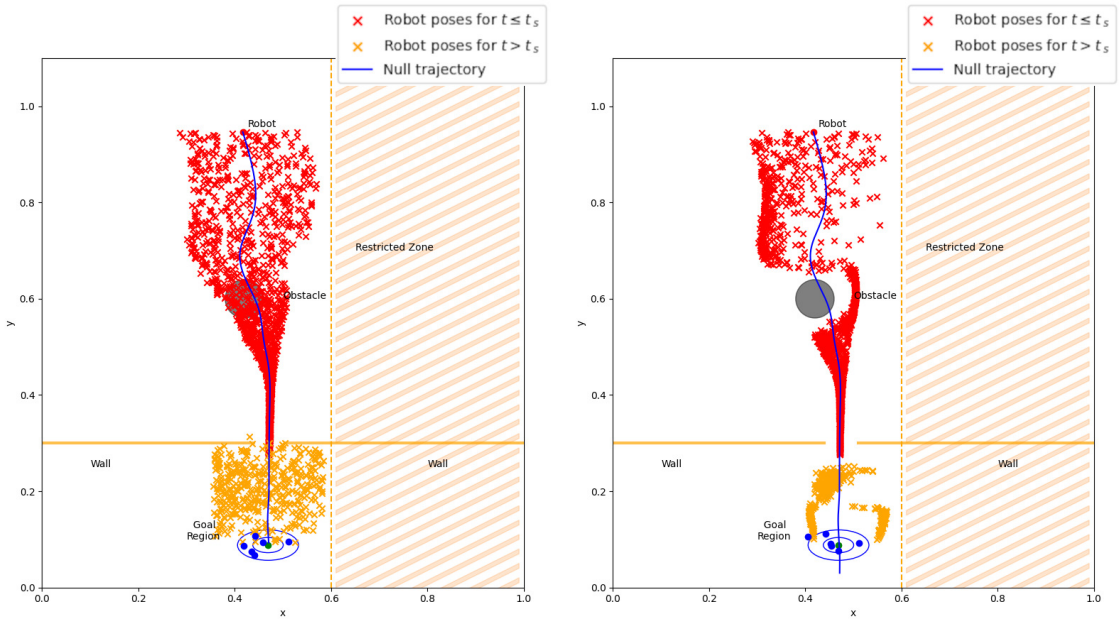
$$\mathbb{E}_{LC}(\eta(s)) = k^* A' \mu + k^* A' \Sigma' \bar{G} \alpha,$$

$$\mathbb{E}_{NS}(\eta(s)) = \frac{\beta}{\gamma} (\hat{k}^* - k^* A' \hat{K}) \pi(\xi|q).$$

The term $\mathbb{E}_{LC}(\eta(s))$ is not influenced by the RL policy π. In order to correct the null trajectory, in this modification of EU-KGRL, $\mathbb{E}_{LC}(\eta(s))$ is computed with the original reference trajectory T_r obtained from the demonstrations, and becomes:

$$\mathbb{E}_{LC}(\eta(s)) = k^* A \mu + k^* A \Sigma \bar{G} \alpha,$$

with $A = (K + \lambda \Sigma)^{-1}$, as defined in Chapter 3.



(a) Corrected EU-KGRL: Episodes 0-5

(b) Corrected EU-KGRL: Episodes 250-255

Figure 6.4: Exploration behavior of EU-KGRL with null trajectory correction over five episodes, before (a) and after training (b) for 250 episodes. The blue dots show the sampled goal positions g, the green dot denotes μ_g.

This change makes $\mathbb{E}_{LC}(\eta(s))$ equivalent to the LC-KMP formulation in [9]. For $\mathbb{E}_{NS}(\eta(s))$, which encodes the null-space projector, the modified reference trajectory T_r' from Equation 5.1 is used. This way, the part of the LC-NS-KMP formulation that is not influenced by the null-space action ξ is computed using the information from the demonstrations, and therefore not affected by any external variance. However, the null-space projector does modulate the LC-KMP trajectory according to the external variance.

This behavior can be seen in Figure 6.4. The null trajectory is equivalent to the KGRL null trajectory from Figure 6.1, and the positional offset for the exploration at $t > t_s$ is removed. The modified EU-KGRL version with trajectory correction still manages to learn to reach the target while avoiding the obstacle. However, no improvement in training performance can be observed. This is shown in Figure 6.5, which compares the training rewards of EU-KGRL with and without null trajectory correction. In every metric, both versions show similar performance.

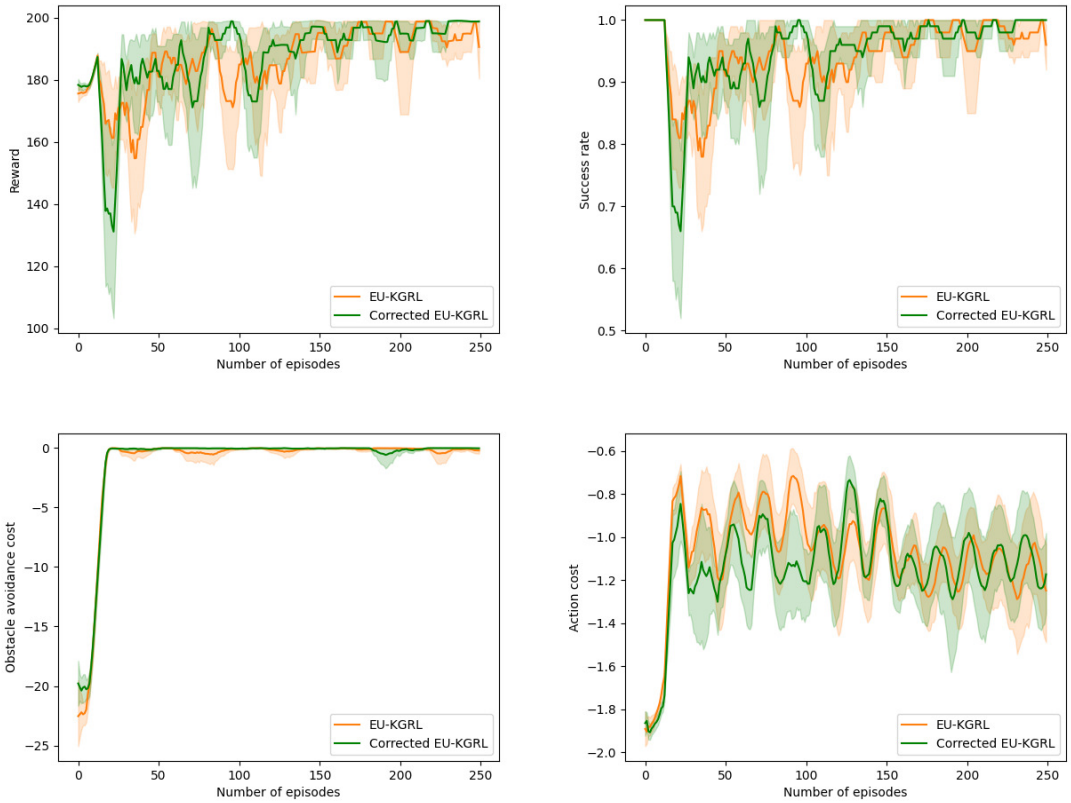


Figure 6.5: The performance of EU-KGRL with and without correction of the null trajectory over 250 training episodes. Each algorithm was run ten times. Shown are four evaluation metrics, the overall reward (top left), success rate (top right), obstacle avoidance cost (bottom left), and action cost (bottom right), each plotted against the number of episodes. The solid lines represent the mean values, while the shaded ares indicate the 95% confidence intervals with a moving average window of size ten. Both versions show similar performance and learn to solve the primary task together with obstacle avoidance.

6.2 Pose Estimation Accuracy

Before conducting experiments on the robot in the real world, the position accuracy of the camera pose estimation pipeline was analyzed. In the experiments presented in this thesis, only the target position is estimated, as detailed in Chapter 1.2. It is assumed that the true orientation of the target is known. Consequently, orientation is not addressed. In the following, the name *pose estimator* will be used to refer to the pose estimation pipeline implemented in Chapter 5.1, even if only the position output is used.

Collection of Ground Truth

In order to evaluate the accuracy of the pose estimator, the corresponding position of the target object in the real world has to be known. For this thesis, this was done using the forward kinematics of the SARA robot. A Python script was written that implements the ground truth recording procedure. As a first step, the robot picks up the Ethernet connector. The robot switches into gravity-compensation mode, and the user moves the robot's end-effector towards the Raspberry Pi (which is fixed to the workbench) and plugs the connector in (see Figure 6.6). The user confirms the pose by pressing a button on the external input device on the workbench. The *pysara* library returns the recorded position using forward kinematics. The user can then move the Raspberry Pi to another position on the workbench and repeat the process.

As described in 5.1, the pose estimation pipeline returns the pose (in this case, position) of the Ethernet connector port opening. Therefore, the collection of ground truth positions has to be done with respect to the port opening as well. This is done by configuring the Tool Center Point (TCP) for the *pysara* library in such a way that it aligns with the opening of the port when the Ethernet connector is plugged in with the robot. The required transformation between the robot's end link and the TCP was determined by measuring the x-, y-, and z-distances from the gripper's base to the port opening with a ruler, while the robot held the connector in place.

It should be noted that this method of recording ground truth data is imperfect. Potential inaccuracies from the robot's forward kinematics can influence the measurements. Yet, it was considered the most accurate means of data collection available for this thesis. This problem is also acknowledged in [2], where a pose estimator for electrical connector sockets was evaluated, while using the pose obtained from an industrial robot. Unfortunately, no comprehensive study about the positional accuracy of the SARA robot existed at the time of writing. Another potential source of inaccuracies in the described ground truth recording method is the use of ruler measurements to configure the TCP. However, offsets resulting from an incorrect TCP configuration should remain consistent across all robot configurations. This makes it possible to correct them manually, which will be discussed later in this section. For now, the imperfect measurements are used, as they represent the best available option. For simplicity, they are still referred to as the *ground truth*, even if the actual, true pose cannot be determined independently of the robot setup.

As shown in Figure 6.6, ten poses were recorded, with a distance of 16 cm in the x- and y-direction towards each other. This distance was chosen because the metal rails that

secure the Raspberry Pi to the workbench are also spaced 16 cm apart. It should be noted that the robot is not perfectly calibrated towards the workbench. This means that, while the true z-position of the target object remains the same relative to the workbench across the ten test positions, the same is not true for the measured positions in the robot's base frame.

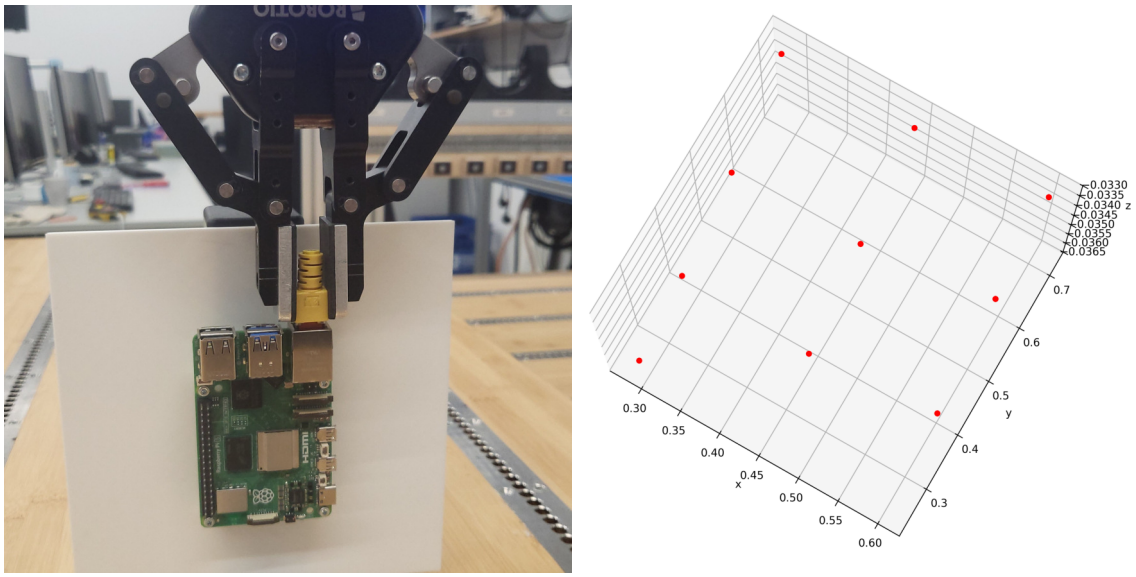


Figure 6.6: For the recording of the ground truth, a user plugs the Ethernet connector into the device with the robot in gravity-compensation mode (left image). The positions are obtained from the robot's forward kinematics, and shown in the right image. They are presented in the robot's base frame in meters.

Evaluation Metric

For the evaluation, three metrics are defined: the Mean Position Error (MPE), the standard deviation of estimates ($\tilde{\sigma}$), and the maximum distance to the mean estimate (d_{max}). Let $\tilde{g}_n^e \in \mathbb{R}^3$ denote one of N position estimates, and $g^{tr} \in \mathbb{R}^3$ the corresponding true target position obtained by using the procedure described above. MPE, $\tilde{\sigma}$ and d_{max} are then defined as:

$$MPE = \frac{1}{N}\sum_{n=1}^{N} \|\tilde{g}_n^e - g^{tr}\|, \tag{6.5}$$

$$\tilde{\sigma} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} (\|\tilde{g}_n^e - \tilde{\mu}^e\|)^2}, \quad \text{with } \tilde{\mu}^e = \frac{1}{N}\sum_{n=1}^{N} \tilde{g}_n^e, \tag{6.6}$$

$$d_{max} = \max_n \|\tilde{g}_n^e - \tilde{\mu}^e\| \tag{6.7}$$

where $\|\cdot\|$ is the Euclidian norm and $\tilde{\mu}^e$ is the average position of all estimates. The three values are calculated for each of the ten different test positions, with N=50 position estimates for every test position.

Evaluation using the Intel RealSense Camera

The pose estimation pipeline was first configured with the Intel RealSense camera. 50 position estimates were obtained for each of the ten test positions. For each estimate, a new camera frame was recorded to take image noise into account. Plots of all 500 position estimates, together with their respective ground truths and the covariances, are provided in Appendix A. The values for the MPE, $\tilde{\sigma}$, and d_{max} are presented in Table 6.7. Additionally, the distance between the ground truth g^{tr} and the position of the camera is given, and denoted as d_{cam}. The position of the camera (in robot base frame) was taken from the calibration file, and d_{cam} was obtained by calculating the Euclidean distance between the camera position and g^{tr}.

	MPE (mm)	$\tilde{\sigma}$ (mm)	d_{max} (mm)	d_{cam} (mm)
Test position 1	3.1892	0.5568	1.0838	288.5314
Test position 2	8.5617	0.3080	0.9577	327.4588
Test position 3	4.3205	0.6860	1.9235	426.0787
Test position 4	3.1713	0.6389	1.8363	448.9278
Test position 5	7.9876	0.5178	1.0736	475.0525
Test position 6	9.3909	1.0112	2.7096	548.4802
Test position 7	3.6468	0.4191	0.7511	609.4972
Test position 8	6.9590	1.1639	3.0240	628.5869
Test position 9	11.0169	1.3198	3.5521	685.4295
Test position 10	4.0523	0.6197	1.3962	749.7836

Table 6.7: MPE, $\tilde{\sigma}$, d_{max}, and d_{cam} for each of the ten test positions. For every position, 50 estimates were made using the Intel RealSense camera. For every test position, the MPE is higher than $\tilde{\sigma}$, and even higher than d_{max}.

As shown in Table 6.7, the pose estimator achieves millimeter-level accuracy. Depending on the test position, the MPE ranges from around 3 mm to 11 mm. The standard deviation $\tilde{\sigma}$ is in the range of around 0.3 mm to 1.3 mm, while the maximum distance d_{max} to the mean estimate $\tilde{\mu}^e$ is around 0.7 mm to 3.5 mm. While a Mean Position Error of only a few millimeters is promising, it should be noted that the MPE is substantially higher than $\tilde{\sigma}$ and even d_{max} for every single test position. This means that the estimates are closer to each other than they are to the ground truth, as every observed estimate is closer to the mean prediction $\tilde{\mu}^e$, than it is to the true position g^{tr}. Thus, the precision of the pose estimation pipeline is higher than the accuracy. This is clearly illustrated in Appendix A where the figures show that for every test position, the ground truth lies outside the distribution of the estimates.

Evaluation using the Intel RealSense Camera with Pose Correction

Compared to the precision, the accuracy of the position estimates is low. There are many possible reasons for this, for example, an imprecise camera calibration. Additionally, as noted above, the recording of the ground truth data inevitably introduces inaccuracies. In an attempt to improve accuracy, the pose estimation pipeline was evaluated with extrinsic pose correction. The idea is that the bias from a potential error in the TCP configuration,

which might have been caused by the use of ruler measurements, can be corrected by applying a transformation to each new pose estimate that accounts for this error. Other pose-invariant biases, for example, caused by inaccurate *extrinsic* camera parameters, could be corrected like this as well.

To compute such a bias correction, a corrective transformation was calculated that minimizes the deviation between the previously obtained pose estimates and their corresponding ground truths. This was done by using the Kabsch-Umeyama Algorithm [54], [55], [56] with a scaling factor of 1; the implementation for this was taken from [57]. The Kabsch-Umeyama Algorithm was applied to the set of mean predictions $\tilde{\mu}^e$ and the set of ground truths g^{tr} for all ten training poses, which returns a 6D transformation that aims to align the two sets. This transformation was then applied to all 500 recorded position estimates, and the MPE, $\tilde{\sigma}$, and d_{max} were calculated again. While it would be necessary to take new position estimates to get independent and more representative results, the aim of this test is to give a brief, qualitative analysis in order to identify constant biases across all training positions.

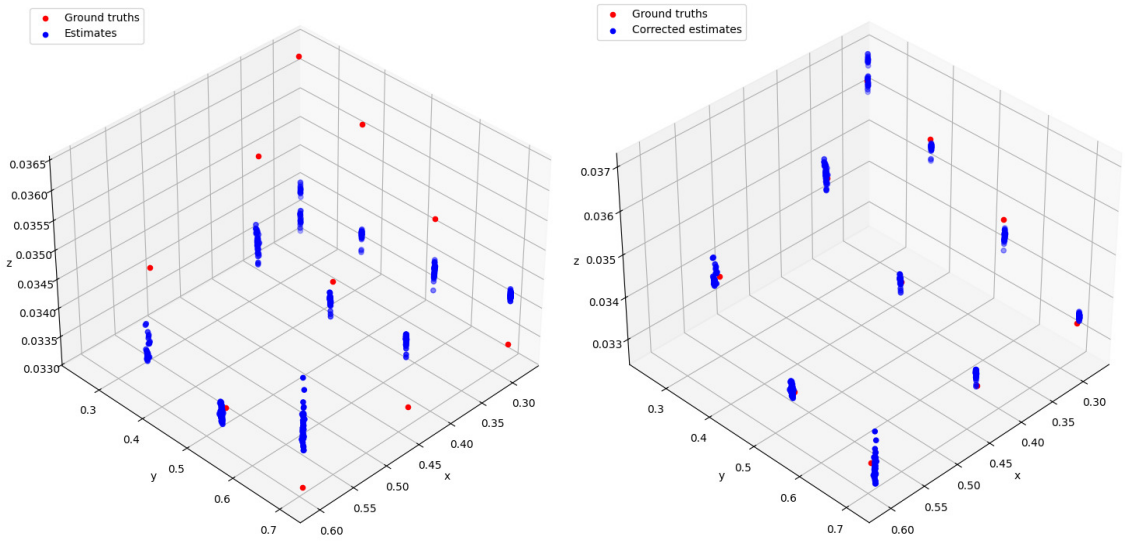


Figure 6.8: The result of applying the pose correction to all 500 position estimates that were taken with the Intel RealSense camera. The left image shows the uncorrected estimates, they do not align well with the ground truth. The right image shows the estimates after a corrective transformation was applied. The means of the predictions align much closer with the ground truths. All positions are given in robot base frame and in meters.

Figure 6.8 shows the result of this correction. The corrected position estimates are visibly closer to their respective ground truths. The values for MPE, $\tilde{\sigma}$, and d_{max} for the corrected estimates are listed in Table 6.9. As expected, $\tilde{\sigma}$ and d_{max} are unaffected by the bias correction. Compared to the results from the uncorrected estimates in Figure 6.7, the MPE for the corrected estimates is lower, achieving a range of around 1.9 mm to 6.1 mm. While this shows an improvement in accuracy, the MPE is still higher than d_{max} in most cases, indicating that the precision remains higher than the accuracy.

	Corrected MPE (mm)	$\tilde{\sigma}$ (mm)	d_{max} (mm)	d_{cam} (mm)
Test position 1	2.5996	0.5568	1.0838	288.5314
Test position 2	3.5219	0.3080	0.9577	327.4588
Test position 3	4.6969	0.6860	1.9235	426.0787
Test position 4	2.9052	0.6389	1.8363	448.9278
Test position 5	2.7164	0.5178	1.0736	475.0525
Test position 6	3.0303	1.0112	2.7096	548.4802
Test position 7	2.9827	0.4191	0.7511	609.4972
Test position 8	1.8976	1.1639	3.0240	628.5869
Test position 9	6.0935	1.3198	3.5521	685.4295
Test position 10	4.2412	0.6197	1.3962	749.7836

Table 6.9: Corrected values for MPE, $\tilde{\sigma}$, and d_{max} for each of the ten test positions. For every position, 50 estimates were made using the Intel RealSense camera. Each estimate was corrected using a transformation that was obtained with the Kabsch-Umeyama Algorithm.

Evaluation using the Microsoft Azure Kinect Camera

To further improve position estimate accuracy for the test setup, the pose estimation pipeline was configured with the Microsoft Azure Kinect camera, which provides higher-resolution images. Furthermore, ten new test positions were recorded that are closer to the camera and to each other. They were recorded with the Raspberry Pi fixed to ten neighboring clamp holes on the same metal rail on the workbench, where the holes are spaced 1 cm apart. The idea behind this change is to reduce the effects of an inaccurate *intrinsic* camera calibration. While the pose estimation pipeline tries to undistort the camera image, the process relies on the accuracy of the intrinsic camera parameters. Assuming that the effects of the remaining image distortion are similar in regions of the camera frame that are close to each other, the effect of the distortion on the position estimates should be comparable across the ten test positions when they are closer together[16]. The idea is to now treat this distortion effect, which presumably exhibits similar characteristics across all ten test positions, as another constant bias that can be accounted for using the correction method described above. While this assumption oversimplifies the problem of camera image distortion, the goal is to test the pose estimator on ten test positions that are close to each other, so that all camera images exhibit similar characteristics.

Again, 50 position estimates for each of the ten test positions were made, where a new camera image was taken for every position estimate. The estimates were corrected using the transformation that was obtained from the Kabsch-Umeyama Algorithm, to account for biases. The corrected and uncorrected position estimates are shown in Figure 6.10.

[16] This assumption is based on the fact that lens distortion is typically modeled as a smooth spatial function of image coordinates; consequently, distortion effects vary gradually across the image, and neighboring pixels experience similar distortion [33] [34].

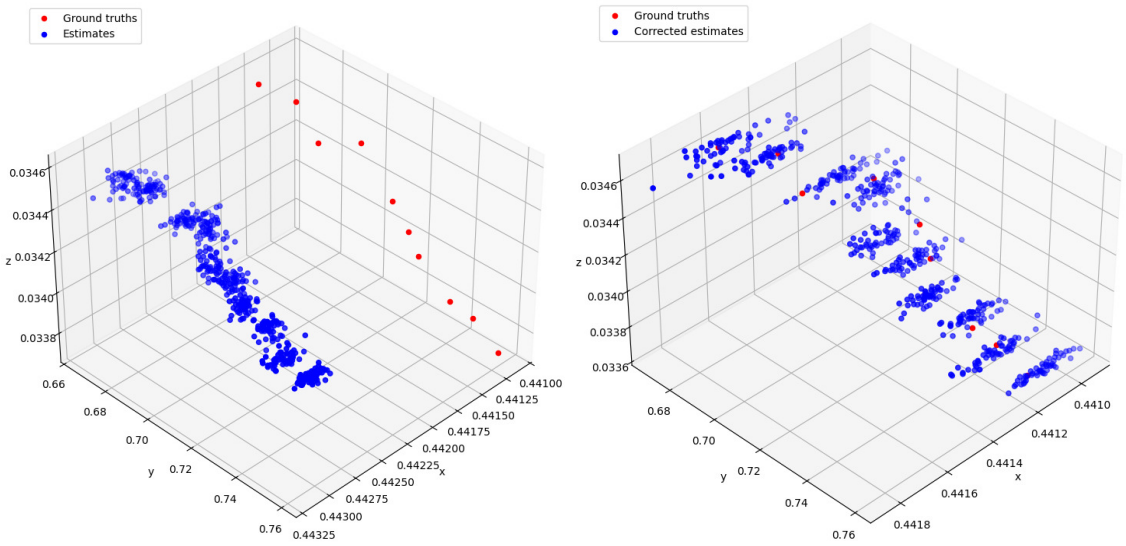


Figure 6.10: The result of applying the pose correction to all 500 position estimates that were taken with the Microsoft Azure Kinect camera. The left image shows the uncorrected estimates, which do not align well with the ground truth. The right image shows the estimates after a corrective transformation was applied, the means of the predictions align much closer with the ground truths. All positions are given in robot base frame and in meters.

	MPE (mm)	Corrected MPE (mm)	$\tilde{\sigma}$ (mm)	d_{max} (mm)	d_{cam} (mm)
Test position 1	4.2042	0.1686	0.1724	0.3381	292.4229
Test position 2	4.3196	0.2378	0.2347	0.4729	301.7705
Test position 3	4.2427	0.1831	0.1784	0.4391	311.2260
Test position 4	4.2207	0.1778	0.1733	0.3712	320.6758
Test position 5	4.2743	0.2449	0.2156	0.5512	329.9902
Test position 6	4.0826	0.2628	0.2042	0.5542	339.3658
Test position 7	4.0125	0.1777	0.1900	0.5340	348.7219
Test position 8	3.7340	0.3177	0.2177	0.5772	358.4431
Test position 9	4.0517	0.1922	0.2325	0.6466	368.1675
Test position 10	3.7590	0.2978	0.2781	0.6057	377.8000

Table 6.11: Values for MPE, corrected MPE, $\tilde{\sigma}$, d_{max}, and d_{cam} for each of the ten test positions. For every position, 50 estimates were made using the Microsoft Azure Kinect camera. Each estimate was corrected using a transformation that was obtained with the Kabsch-Umeyama Algorithm. The MPE was calculated for the original estimates and for the corrected estimates.

The values for $\tilde{\sigma}$ and d_{max} are presented in Table 6.11, together with the MPE for the corrected and uncorrected position estimates. The uncorrected MPE for this setup lies in a small range of around 3.7 mm to 4.3 mm. After correction, the MPE has a value of under 0.3 mm for every test pose. This is consistently smaller than d_{max}, and even smaller than $\tilde{\sigma}$ for some test poses, indicating a much higher accuracy than achieved in the previous

setup. This is shown in the plots of the corrected position estimates in Appendix B, which show that the distribution of position estimates is close to the ground truth. There are many factors that could potentially contribute to this increase in accuracy, for example, the higher image resolution, the closer distance between the target and the camera, or the fact that the ten test positions are closer together. However, a comprehensive study about all the effects of the setup on position estimate accuracy is beyond the scope of this thesis. Overall, the test results show that the position estimates can be improved with a bias to achieve sub-millimeter accuracy, at least when the target object remains in a small region.

Additional Observations

Based on the distribution of position estimates, it was observed that the variance is not the same in all directions. The distribution has the shape of an elongated ellipse that is aligned with the camera viewing direction. This can be seen in Figure 6.12, where a line is drawn from the camera's position to the mean of the position estimates (the plot was zoomed in to show the estimates up close). The spread of the estimates roughly follows this line. This effect was observed for every test position in all setups, suggesting that most of the variance in the position estimates stems from uncertainty in the depth estimation relative to the camera.

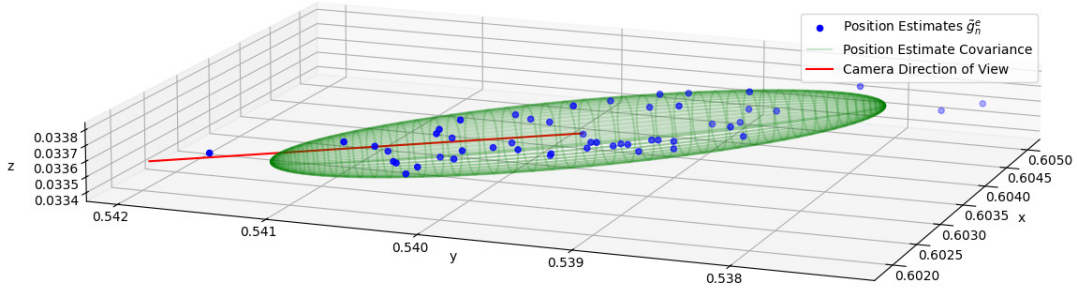


Figure 6.12: 50 position estimates from the pose estimation pipeline, which was configured with the Intel RealSense camera. A line was drawn from the camera's position to the mean of the estimates, and the plot was zoomed in to show the estimates up close. This line represents the viewing direction of the camera (towards the estimated position of the target object). The spread of the estimates seems to roughly follow this line. All positions are given in robot base frame and in meters.

These observations imply that the covariance of the position estimates provides information about the precision of the estimation, as well as the directions in which the precision is higher or lower. This is valuable information for EU-KGRL, as the algorithm is able to shape the exploration space of the robot according to an externally-provided covariance matrix. However, only using the covariance obtained from the position estimates on the same test position would not result in the construction of an ideal exploration space when combined with an estimate from the pose estimation pipeline. The accuracy of position estimates is considerably lower than the precision, meaning that the constructed exploration space might end up being too far away from the actual target position to solve the proposed task of connector-insertion.

In order to solve this problem, the effects on the position estimate error that are not caused by the pose estimation pipeline (for example, camera calibration errors) have to be modeled into the covariance as well. However, this is not trivial. While it was attempted to analyze the position estimates over multiple test positions, to construct a covariance that would include multiple sources of uncertainty, the resulting error distribution was large and did not follow a specific direction like the one shown in Figure 6.12. Therefore, it was not considered suitable for shaping the exploration space of EU-KGRL. For the real-world experiments conducted in this thesis, only the covariance of the pose estimation pipeline was used, which was obtained from 50 position estimates of the same target position. The estimates are manually corrected so that the mean of the 50 estimates aligns with the ground truth. Consequently, EU-KGRL learns to compensate for the imprecision of position estimates, but doesn't take inaccuracy into account.

6.3 Experiments on the Real Robot

For the experiments in the real world, the robot setup described in Chapter 4.3 was used. As in the original KGRL experiment setup [8], five demonstration trajectories $D = \{\{t_{n,m}, p_{n,m}\}_{n=1}^{400}\}_{m=1}^{5}$ of successful connector-insertion were recorded using kinesthetic teaching. The input t is the current time step, and the output $p = \begin{bmatrix} x & y & z & a_z & a_y & a_x \end{bmatrix}^T$ (with Euler angles a_x, a_y, and a_z) is the 6D pose of the robot end-effector in target frame. The target frame for each demonstration is given by the last recorded pose in the trajectory, as it corresponds to the desired configuration of an inserted connector. These design decisions were adopted from [8].

For demonstrating the trajectories, as shown in Figure 6.13, a software tool that was developed in [58] was used. The recorded trajectories are plotted in Figure 6.14, alongside the fitted Gaussian Mixture Model derived from the demonstrations. Note how Figure 6.14 shows that the variance in the demonstrations becomes smaller close to the target frame.

KGRL and EU-KGRL are configured with an RL policy $\pi(\xi|q_t)$ that generates null-space actions $\xi \in \mathbb{R}^6$ based on the state vector q_t:

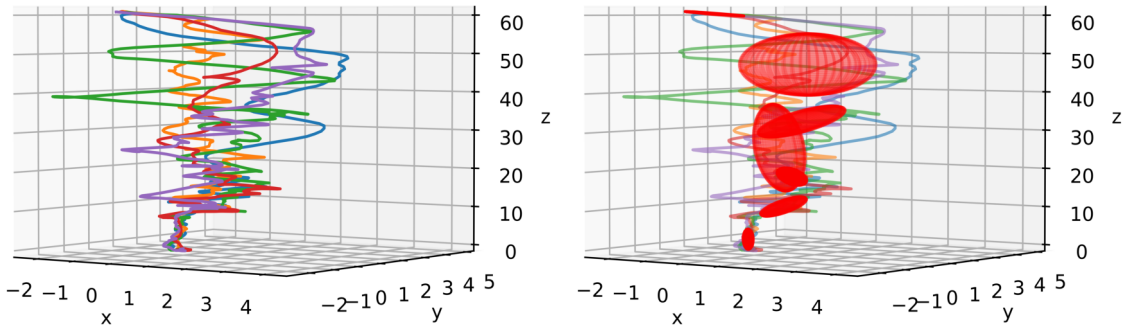
$$q_t = \begin{bmatrix} t & \tilde{p}_t^T & f_t^T & \xi_{t-1}^T \end{bmatrix}^T$$

where t is the current time step, $\tilde{p}_t$ is the current 6D robot end-effector pose in expected target frame. f_t is the 6D wrench measured at the center of compliance, and ξ_{t-1} is the null-space action from the previous time step (initialized with $\xi_0 = 0$).

Similar to the simulation experiments, the RL policy $\pi(\xi|q_t)$ was configured as a DNN consisting of 2 hidden layers with 256 neurons each. Training was done in Stable-Baselines3 [51] using TQC [52] [53] configured with the following parameters:

```
learning rate = 0.001,
soft update coefficient = 0.01,
discount factor = 0.995,
training frequency = 8,
gradient steps = 8,
entropy regularization coefficient = auto with initial value of 0.1
```

Figure 6.13: A user giving a demonstration of inserting an Ethernet connector using kinesthetic teaching. For recording the demonstrations, a software tool that was developed in [58] was used, which can be controlled using the input device that is placed on the workbench.



(a) Five demonstrated trajectories [mm]

(b) GMM from user demonstrations [mm]

Figure 6.14: Five demonstrations given by the user (a). The demonstrations are shown in mm in target frame. A fitted GMM with seven Gaussian components (as ellipses with a standard deviation of 2) is shown in (b). Close to the target frame ([0, 0, 0]), the demonstrations are closer together, and the covariance of the Gaussian component is small. Hence, the variance becomes small close to the target frame.

The reward function r_t is defined as follows:

$$r_t = r_a + r_f + r_T, \tag{6.8}$$

$$r_a = -0.01 \xi_t^T \xi, \tag{6.9}$$

$$r_f = -0.01 f_t^T f_t, \tag{6.10}$$

$$r_T = \begin{cases} 60 & \text{at terminal step T if successful,} \\ -50 & \text{if robot detects collision,} \\ 0 & \text{otherwise,} \end{cases} \tag{6.11}$$

where r_a is the action cost, r_f is the interaction force cost and r_T is the terminal reward. An episode is considered successful if the Euclidean distance between the end-effector and the true target pose is smaller than a predefined threshold, or when the user manually labels it as successful with the input device. While this does require the ground truth to be known, it is still an improvement over the original setup in [8], where the user had to label each successful episode manually. In a real-world scenario with an intact cable, this could be addressed by verifying the presence of an electrical signal on the cable. Collisions are detected using SARA's collision detection feature, but can also be triggered manually using the input device.

Again, this configuration is similar to the one from the original KMP experiment in [8], but with a modified state vector q_t. The 6D pose of the robot end-effector in target frame is replaced with $\tilde{p}_t$, the 6D pose in estimated target pose. Additionally, q_t was extended with the null-space action from the previous time step. The idea behind this extension is to provide the RL agent with enough information to solve the task. In every episode, $\tilde{p}_t$ might deviate from the actual position p_t in target frame in a different way, meaning that the RL agent does not have reliable information about the robot's end-effector position. The robot should respond to contact forces in a way that minimizes them, and should avoid large movements after it navigated the connector into the port entrance. In order to do this, it is necessary for the policy to know which null-space action it produced previously, to output an action that is more or less similar to produce smaller or bigger movements.

For both KGRL and EU-KGRL, the LC-NS-KMP generates a new robot position $\tilde{p}_t$ for every time step t, while using the null-space action ξ from the RL policy $\pi(\xi|q_t)$. The LC-NS-KMP is constructed from the five recorded demonstrations. In the original KGRL experiment in [8], the LC-NS-KMP was constrained to outputs that have a positional offset of less than 2 mm from the target frame in the XY-plane. For the experiments in this thesis, the linear constraints were relaxed to 2 cm in the XY-plane with respect to the estimated target frame, resulting in the following parameters for the construction of EU-KGRL[17]:

$$\begin{aligned} g_{n,1}^\top &= [1,0,0,0,0,0], \quad c_{n,1} = -0.02, \\ g_{n,2}^\top &= [-1,0,0,0,0,0], \quad c_{n,2} = -0.02, \\ g_{n,3}^\top &= [0,1,0,0,0,0], \quad c_{n,3} = -0.02, \\ g_{n,4}^\top &= [0,-1,0,0,0,0], \quad c_{n,4} = -0.02, \quad \forall n = 1 \dots N. \end{aligned}$$

[17] Here, $g_{n,f}$ represents a linear constraint vector following the notation in [8], not to be confused with the target pose g.

A new target goal estimate $\tilde{g}$ in robot base frame[18] is obtained before the start of every episode. The end-effector pose $\tilde{p}_t$ is given in reference to this frame. To compare KGRL and EU-KGRL, both RL algorithms were trained five times, with 50 episodes for each run in every experiment.

Following the formulation in Chapter 5.3, EU-KGRL was implemented using these parameters:

$$
\begin{aligned}
t_s &= 200, \\
\Sigma_E &= \tilde{\Sigma}_g, \\
C_{mod} &= 40.
\end{aligned}
$$

The external covariance Σ_E is set to the covariance constructed with the camera pose estimator. t_s is set to 200, which is half of the number of time steps in every demonstration. For C_{mod}, the value from the simulation experiments was reused, as it yielded good results in the real-world experiments as well. EU-KGRL was tested with and without null trajectory correction, but no difference in performance was observed. The following results are from experiments without trajectory correction.

Experiment R1 - Artificial Pose Estimation

As a first experiment in a simplified scenario, the described setup was tested without the camera pose estimator. Instead, five artificial pose estimates $\tilde{g}$ were defined, representing the true target pose with an offset Δx of 0mm, $\pm$1mm, and $\pm$2mm in the x-direction:

$$
\tilde{g} = \begin{pmatrix} 0 & 1 & 0 & 0.4428 + \Delta x \\ 0 & 0 & 1 & 0.8496 \\ 1 & 0 & 0 & 0.3800 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \Delta x = \begin{cases} 0 & \text{for } k \bmod 5 = 0, \\ 0.001 & \text{for } k \bmod 5 = 1, \\ -0.001 & \text{for } k \bmod 5 = 2, \\ 0.002 & \text{for } k \bmod 5 = 3, \\ -0.002 & \text{otherwise,} \end{cases} \tag{6.12}
$$

where k is the current RL training episode. When the offset Δx is zero, $\tilde{g}$ is equivalent to the exact pose of the connector port, which was determined by taking the last end-effector pose from one of the demonstrations. This way, the estimated target is equivalent to the true target in the first training episode, and cycles through the list of target offsets during training. For 50 training episodes, the RL agent observes each of the five resulting different target poses ten times.

[18] If not stated otherwise, all following poses and positions are given in robot base frame and in meters.

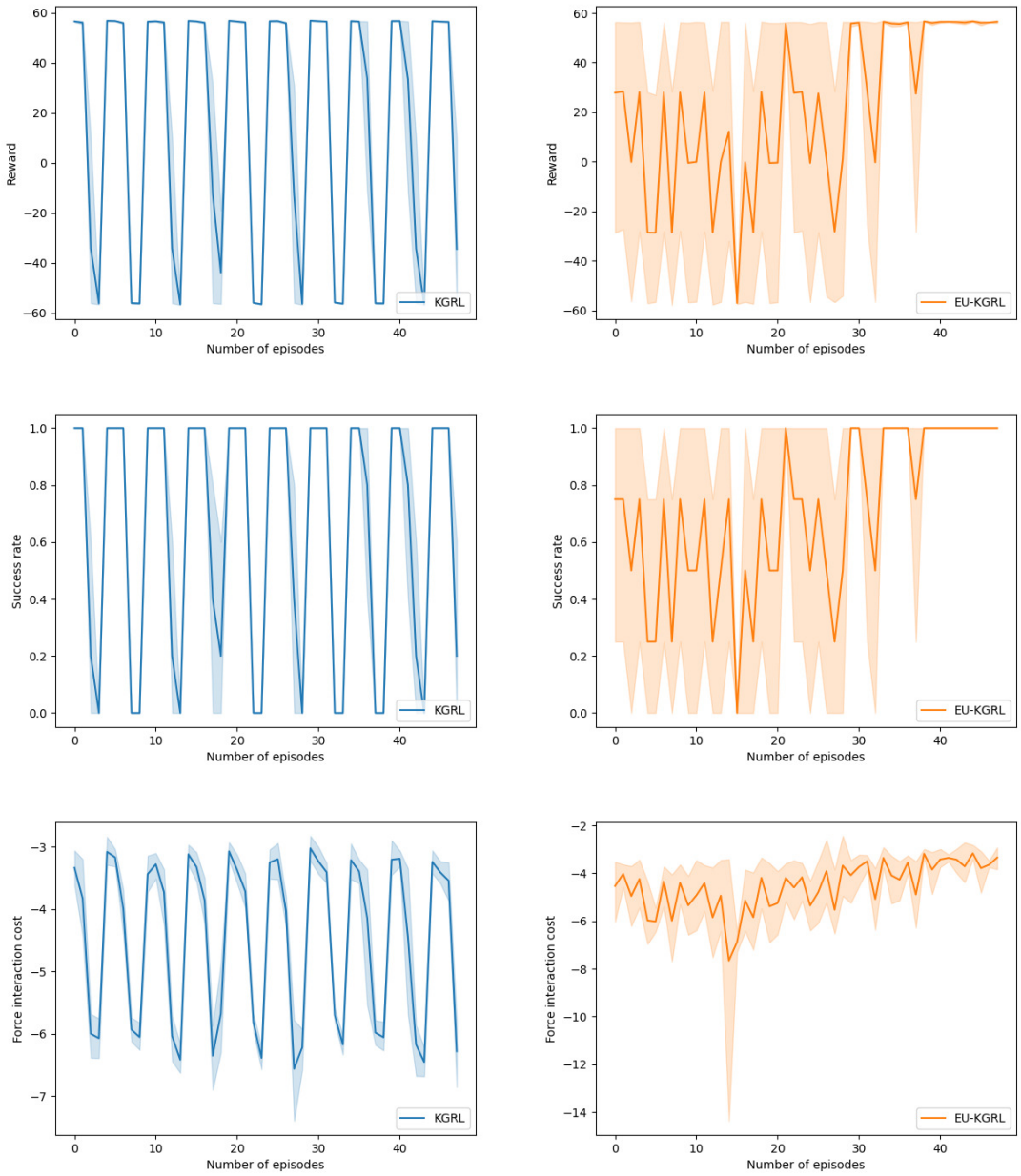


Figure 6.15: The performance of KGRL and EU-KGRL over 50 training episodes for Experiment R1 on the robot. Each algorithm was run five times. Shown are three evaluation metrics, the overall reward (top), success rate (center), and force interaction cost (bottom), each plotted against the number of episodes. The solid lines represent the mean values, while the shaded ares indicate the 95% confidence intervals. While the performance for EU-KGRL converges, KGRL fails to learn the task.

The artificial covariance matrix $\tilde{\Sigma}_g$ for EU-KGRL was designed to have a high variance in the the x-direction, a low variance in the y- and z-directions, and barely any variance for the orientation:

$$\tilde{\Sigma}_g = \begin{pmatrix} 1 \cdot 10^{-3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 \cdot 10^{-4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 \cdot 10^{-4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \cdot 10^{-6} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \cdot 10^{-6} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \cdot 10^{-6} \end{pmatrix}$$

Figure 6.15 shows the results of training KGRL and EU-KGRL for 50 episodes. The total reward (Equation 6.8), success rate, and force interaction cost (Equation 6.10) for every episode are given. As the terminal reward outweighs the force interaction cost and the action cost, the plots for the total reward and for the success rate look similar.

KGRL does not manage to learn the task of connector-insertion reliably. The plots for the total reward and for the success rate show that KGRL successfully solves the task across three consecutive episodes. However, this is then followed by two unsuccessful episodes. This behavior can be explained by how the artificial target pose estimate is generated in Equation 6.12. In unsuccessful episodes, the estimate has an offset Δx of ± 2 mm to the true target position. An offset of ± 1 mm, however, is compensated by KGRL. An episode is terminated unsuccessfully when the robot observes contact force readings that exceed the threshold for the collision detector. The force interaction cost is generally higher in these episodes. This can also be seen in Figure 6.15, where the plot of the force interaction cost exhibits similar trends as the plots of the reward function and the success rate.

EU-KGRL learns to reliably solve the task in under 40 episodes, and offsets of ± 2mm can be compensated for. Additionally, an improvement in the mean force interaction cost can be observed, indicating that EU-KGRL learns to minimize contact forces.

Experiment R2 - Camera Pose Estimation using the Intel RealSense Camera

In the second experiment, the pose estimation pipeline is used, which was configured with the Intel RealSense camera. As mentioned in Chapter 1.2, the problem is reduced to estimating the position of the target. Hence, the target estimate $\tilde{g}$ for training is defined as follows:

$$\tilde{g} = \begin{pmatrix} 0 & 1 & 0 & \hat{g}_x \\ 0 & 0 & 1 & \hat{g}_y \\ 1 & 0 & 0 & \hat{g}_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.13}$$

where $\hat{g}_x$, $\hat{g}_y$, and $\hat{g}_z$ denote the x-, y-, and z-position of the *corrected* target pose estimate obtained from the camera pose estimation pipeline. The estimate has to be corrected first because the pose estimator has a high precision, but a low accuracy. As discussed in Section 6.2, there are several factors that could cause this (for example, camera calibration errors), which cannot be modeled easily. The constant bias that was calculated in Section 6.2 based on the 500 estimates across ten different test positions was not sufficient to reduce the estimation inaccuracies enough for them to be negligible. Therefore, for this experiment, the mean of the position estimates was corrected to align with the true training position. 50 pose estimates were taken in advance, and the transformation between the mean of

the estimates and the true target pose was calculated. This transformation was then used to correct every new pose estimate, resulting in the values for $\hat{g}_x, \hat{g}_y$, and $\hat{g}_z$.

The external covariance matrix $\tilde{\Sigma}_g$ for EU-KGRL is configured to include the position covariance from the pose estimator:

$$\tilde{\Sigma}_g = \begin{pmatrix} & & & 0 & 0 & 0 \\ & \Sigma_g & & 0 & 0 & 0 \\ & & & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^{-6} & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^{-6} & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^{-6} \end{pmatrix}$$

where $\Sigma_g \in \mathbb{R}^{3x3}$ is the pose estimator covariance. Like in the previous experiment, the orientation variance is set to be small. As described in Chapter 5.1, Σ_g was obtained by constructing the covariance matrix from 50 position estimates that were taken before the start of the training. During training, it is fixed.

EU-KGRL was trained first. To get independent results, a new Σ_g was constructed for every run from 50 new position estimates. For every episode, a new position estimate is obtained using a new camera image. All corrected position estimates for every run were saved during training. Afterward, KGRL was trained on the exact same position estimates. This was done to compare both algorithms under conditions that are as similar as possible.

The training performance of KGRL and EU-KGRL for Experiment R2 is shown in Figure 6.16. Like in Experiment R1, the total reward closely follows the shape of the success rate graph. EU-KGRL manages to obtain a success rate of 1 in under 40 episodes. The success rate of KGRL does not converge, it does not learn to solve the task. In contrast to KGRL, EU-KGRL optimizes the force interaction cost over time, learning to minimize unnecessarily strong contacts with the environment.

The true target pose g for Experiment R2 was:

$$g = \begin{pmatrix} 3.33 \cdot 10^{-3} & 9.99 \cdot 10^{-1} & 1.14 \cdot 10^{-3} & 4.43 \cdot 10^{-1} \\ 4.57 \cdot 10^{-6} & 1.14 \cdot 10^{-3} & 9.99 \cdot 10^{-1} & 3.89 \cdot 10^{-1} \\ 9.99 \cdot 10^{-1} & -3.33 \cdot 10^{-3} & -7.74 \cdot 10^{-1} & 3.62 \cdot 10^{-2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.14}$$

It was obtained by plugging in the connector using kinesthetic teaching and reading the pose using the robot's forward kinematics. According to the camera calibration file, the camera was placed at the position $\begin{pmatrix} 2.89 \cdot 10^{-1} & 9.97 \cdot 10^{-1} & 2.59 \cdot 10^{-2} \end{pmatrix}^T$, meaning that the distance between the camera and the target object was around 0.63m.

As mentioned above, the target estimates obtained from the corrected estimator poses were logged during training. The Mean Positional Error (MPE) of all 50 training targets $\tilde{g}$ towards the ground truth g for each of the 5 training runs is shown in Table 6.17. The Maximum Positional Error (MaxPE), meaning the maximum observed distance between $\tilde{g}$ and g, is shown as well (MaxPE $= \max_n \|\tilde{g} - g\|$). Plots of the training poses can be seen in Appendix C.

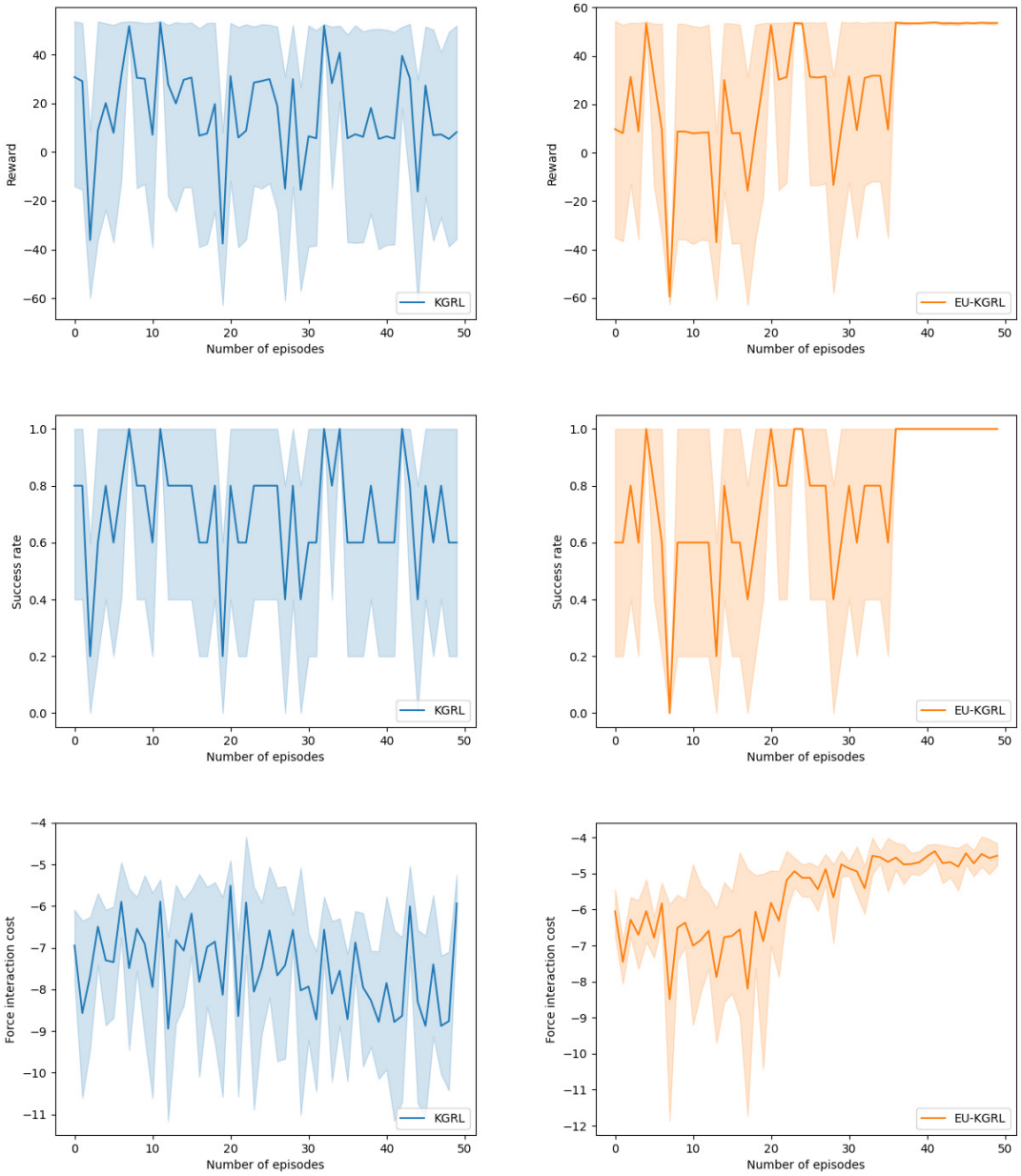


Figure 6.16: The performance of KGRL and EU-KGRL over 50 training episodes for Experiment R2 on the robot. Each algorithm was run five times. Shown are three evaluation metrics, the overall reward (top), success rate (center), and force interaction cost (bottom), each plotted against the number of episodes. The solid lines represent the mean values, while the shaded ares indicate the 95% confidence intervals. While the performance for EU-KGRL converges, KGRL fails to learn the task.

	MPE (mm)	MaxPE (mm)
Training Run 1	0.8715	3.0476
Training Run 2	0.6964	1.8184
Training Run 3	0.5655	1.5406
Training Run 4	0.8646	2.7324
Training Run 5	0.6184	1.7694

Table 6.17: The observed mean (MPE) and maximum (MaxPE) Euclidian distances of all 50 training target poses $\tilde{g}$ towards the ground truth g for every training run.

Experiment R3 - Camera Pose Estimation using the Microsoft Azure Kinect Camera

In a third experiment, the pose estimation pipeline was configured with the Microsoft Azure Kinect camera. $\tilde{g}$ and $\tilde{\Sigma}_g$ were configured like in the previous experiment. However, instead of correcting new pose estimates with a transformation that aligns the mean of the position estimates with the ground truth, the corrective bias estimation across the ten close test positions using the Kabsch-Umeyama Algorithm was performed, as described in Section 6.2. The objective was to compare EU-KGRL and KGRL in a setting in which the ground truth is not known prior to training. Instead, the accuracy of the position estimates is analyzed across multiple test positions in a small region to obtain a corrective transformation that accounts for potential constant offsets. If such a transformation increases the accuracy well enough, so that the distribution of corrected position estimates overlaps with the ground truth [19], the exploration space of EU-KGRL would be shaped in way that allows the robot to reach the target. Within this region, a user would be able to place the target object anywhere, and EU-KGRL would be able to learn the task.

The Raspberry Pi was placed on one of the ten test positions from the Microsoft Azure Kinect setup described in Section 6.2, and both EU-KGRL and KGRL were trained on corrected position estimates over 50 episodes. However, for all ten test positions, both EU-KGRL and KGRL succeeded immediately. The success rate was 1 from the beginning of the training run, and the overall reward neither improved nor decreased. Interestingly, the force interaction cost also did not improve. Apart from a few outliers, this behavior was consistent over multiple runs across all ten test positions. An example is given in Figure 6.18. The overall reward follows the plot of the force interaction cost, as the influence of the interaction cost on the overall reward function is higher than the influence of the action cost[20].

[19] This does happen for the corrected position estimates at most test positions in the Microsoft Azure Kinect evaluation in Section 6.2, and can be seen in Appendix B.

[20] See Equation 6.8; while $\tilde{\zeta}_t$ is defined as $-1 \leq \tilde{\zeta}_t \leq 1$, the readings from f_t can be much higher.

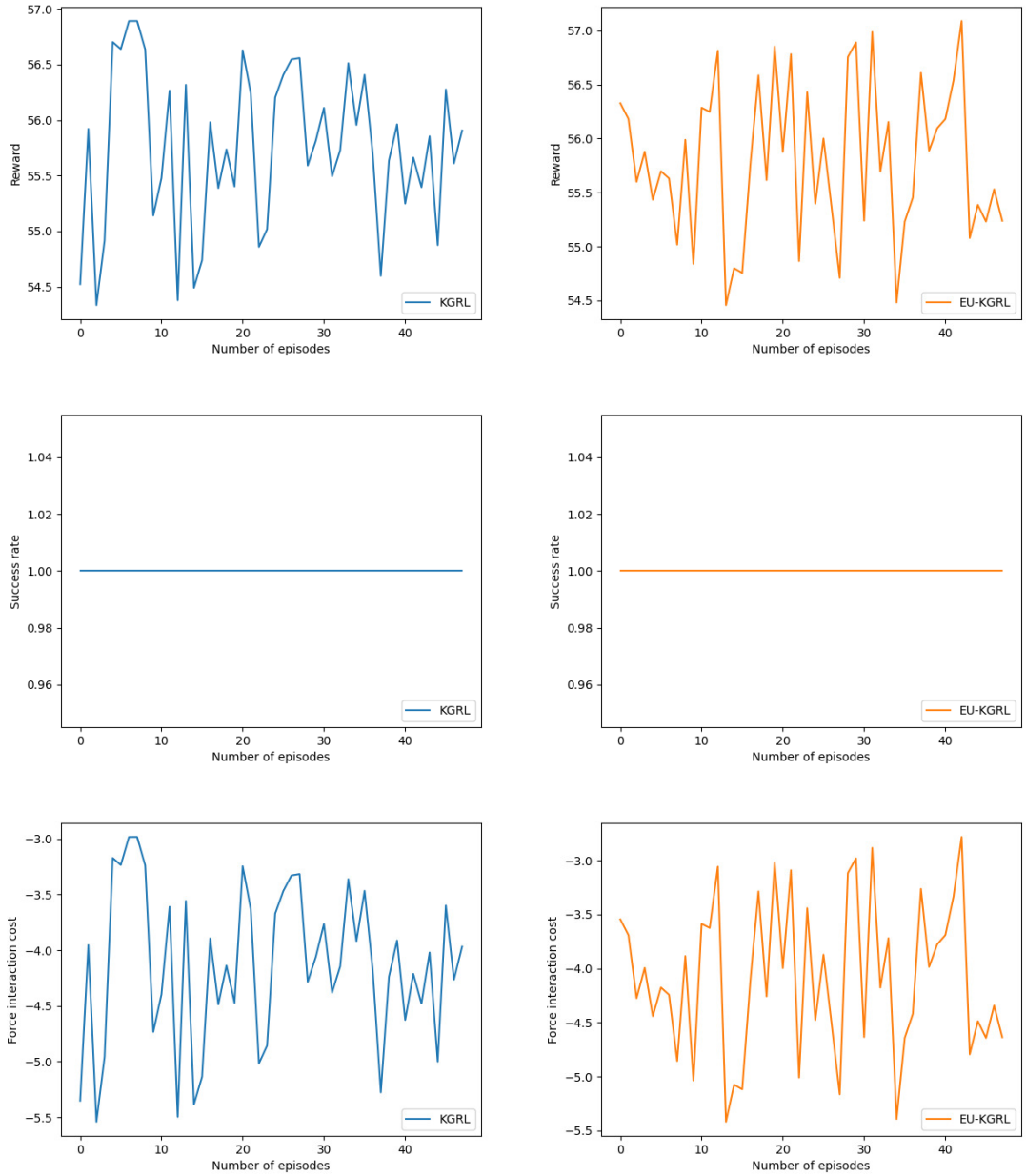


Figure 6.18: An example of the performance of KGRL and EU-KGRL over 50 training episodes for Experiment R3 on the robot. Shown are three evaluation metrics, the overall reward (top), success rate (center), and force interaction cost (bottom), each plotted against the number of episodes. Both algorithms are successful from the first episode.

7

Discussion

7.1 Performance of EU-KGRL compared to KGRL

The results presented in Chapter 6 demonstrate that the proposed extension, External-Uncertainty KGRL (EU-KGRL), can improve performance over KGRL in scenarios where relevant uncertainty is not captured by demonstrations. When external variance is introduced into the system, EU-KGRL is able to exploit information about this variance to guide exploration more effectively.

This is most clearly illustrated in the simulation experiments in Chapter 6. While KGRL and EU-KGRL both successfully learn the secondary objectives (obstacle avoidance and navigating the constrained passage), only EU-KGRL learns to reliably achieve the primary objective of reaching the goal. The low covariance in the demonstrations near the goal region limits exploration in KGRL, preventing it from solving the task.

A key advantage of EU-KGRL is its ability to flexibly adjust the covariance used for exploration. This allows the exploration space to be shaped differently depending on the task requirements, effectively unifying multiple behaviors within a single framework. Depending on the robot's state, EU-KGRL can either use the external covariance or fall back to the standard KGRL formulation, which uses only information from the demonstrations. The conditions for this switch are task-specific and need to be defined by the user. In contrast, for KGRL, the user only needs to give demonstrations, making it the more user-friendly variant, as switching behavior does not need to be defined. However, in certain scenarios, EU-KGRL enables the use of the KGRL framework, where it would otherwise be impossible.

An important thing to consider when introducing external variance to the LC-NS-KMP formulation, which is done for EU-KGRL, is that the generation of the nominal KMP trajectory is affected as well. This can be seen in Figure 6.2, where the null trajectory shows an abrupt offset to the left after EU-KGRL switches to the external variance. Consequently, the exploration space, which is centered around the null trajectory, is shifted as well. The experiments in simulation and on the real robot suggest that this does not necessarily limit the applicability of EU-KGRL, as long as the exploration space remains large enough for the robot to solve the task.

However, in scenarios where it is required to center the exploration space around the mean trajectory obtained from demonstrations, the LC-NS-KMP formulation enables EU-KGRL to be configured accordingly. The LC-NS-KMP formulation separates the generation of the nominal KMP trajectory, the compliance with linear constraints, and the null-space projector into three separate terms [8]. Only recalculating the null-space projector with the modified reference trajectory T_r', and computing the other two terms with the GMR-generated reference trajectory T_r, leads to the generation of a nominal LC-KMP output that is modified based on the external covariance. This idea is similar to the concept of residual RL, which was compared to KGRL in [8], and also combined with DMPs in [13]. In fact, depending on the use case, an algorithm that allows state-dependent switching between KGRL and the residual approach from [8] might deliver performance similar to EU-KGRL. The residual output from the RL policy would need to be scaled according to the external covariance matrix. The benefit of this would be that the exploration space could be aligned directly with the covariance, without the need to tune a hyperparameter (such as C_{mod} in EU-KGRL).

It should also be noted that the simulation environment does not fully capture real-world effects such as contact dynamics. While the simulation provides a useful proof of concept, its results are not directly transferable to physical systems.

7.2 Pose Estimation Pipeline and the Impact on EU-KGRL

The implemented pose estimation pipeline yields promising results. While, for this thesis, each module was trained or configured individually to work with the target device (in this case, a Raspberry Pi), the process could be easily automated. In contrast to approaches that rely on real-world images [2], training on synthetic data only requires the availability of a 3D model and eliminates the need for manual data acquisition and labeling. This is highly attractive for scalable robotic applications and industrial settings.

A comprehensive study on the pose estimator performance under different conditions was beyond the scope of this thesis. However, it should be noted that the pipeline's robustness was remarkably high in the examined setup. With the exception of a few rare outliers, the object detector reliably identified the target object, and the pose estimates were generally consistent. Considering that the object detector and the EagerNet module (which produces the initial unrefined pose estimates) were both trained only on computer-rendered images, this is noteworthy.

As demonstrated in Chapter 6.2, the pose estimation pipeline achieves millimeter-level accuracy in position estimation for target objects positioned between approximately 0.29 m and 0.75 m from the camera. The estimator exhibits high precision, with the standard deviation of the position estimates remaining below one millimeter on average. This indicates that the estimator could be suitable for tasks requiring high repeatability. However, in this context, it should also be considered that changes in lighting conditions can influence the estimates, as the pose estimation pipeline includes a region-based refiner that relies on image statistics. Throughout the experiments in this thesis, lighting conditions remained approximately consistent.

The analysis in Chapter 6.2 shows that the position estimation covariance is not isotropic. Instead, the spread of the estimates typically aligns with the camera's viewing direction. As the covariance matrix for the distribution of the estimates can be empirically obtained by taking multiple measurements, the exploration space of EU-KGRL can be shaped accordingly. A key limitation lies in the accuracy of the estimates. While the estimator exhibits low variance (high precision), its predictions are systematically offset from the true target pose. This discrepancy is particularly problematic for EU-KGRL, as the method relies on an accurate representation of uncertainty. If the covariance does not correctly capture the true error distribution, the resulting exploration behavior may be suboptimal.

Attempts to correct this offset using a pose-invariant transformation derived from local error analysis were only partially successful. While such corrections can improve accuracy in a small region, they do not generalize well across larger workspaces. To investigate camera resolution as a potential limiting factor, the position-estimate analysis for the larger workspace in Chapter 6.2 (with 16 cm spacing between test positions) was repeated using the higher-resolution Microsoft Azure Kinect camera in place of the Intel RealSense. While both accuracy and precision were increased, the problem of a substantially lower accuracy (compared to precision) still persisted. The accuracy error does not appear to be systematic across different test positions when they are spaced farther apart. This could indicate that image distortions may affect position estimates in a way that varies with the target object's location within the camera image. However, many factors could contribute to this inaccuracy. For instance, the recorded ground truth data might be inaccurate due to kinematic errors in the robot.

These findings suggest that achieving the required level of accuracy for EU-KGRL would demand highly precise camera calibration and robot kinematics, which may not be achievable in many real-world settings. An alternative approach is to operate directly in camera frame, eliminating the need for extrinsic camera calibration. By estimating the robot's end-effector pose using the pose estimation pipeline, and using the transformation to the target pose estimate as the new robot pose $\tilde{p}_t$, errors from extrinsic camera calibration and robot kinematics would be eliminated. This idea is similar to the concept of eye-to-hand visual servoing [59], where a robot is observed by an external camera and controlled based on the camera images.

During this thesis, the described approach was implemented and combined with EU-KGRL. As an initial estimate, the imperfect end-effector pose from the robot's forward kinematics was taken. The RBGT tracker [22] [50] (the last step in the pose estimation pipeline) was then used to refine this estimate, based on a publicly available 3D model of the robot's gripper [60]. The initial estimate was good enough for the refiner to determine an accurate pose in camera frame afterward, and eliminated the need for retraining the pose estimation pipeline. An implementation of this can be seen in Figure 7.1. The transformation between the estimated end-effector pose from the refiner and the end-effector pose obtained from forward kinematics was used to correct all subsequent robot poses $\tilde{p}_t$ at each time step in the episode. Additionally, the covariance obtained from end-effector position estimates was added to the external Covariance Σ_E for EU-KGRL. For certain target positions, the approach produced promising results and EU-KGRL did learn to compensate for position estimate uncertainty. However, inconsistent training performance across multiple runs prevented meaningful evaluation.

Future work could include a detailed analysis of gripper pose estimation accuracy, as well as an investigation into the effects of combining multiple external covariances to further explore this extension.

Overall, the results indicate that camera-based pose estimation uncertainty is difficult to implement for EU-KGRL in a real-world setting, primarily because accurately modeling all relevant error sources is challenging.

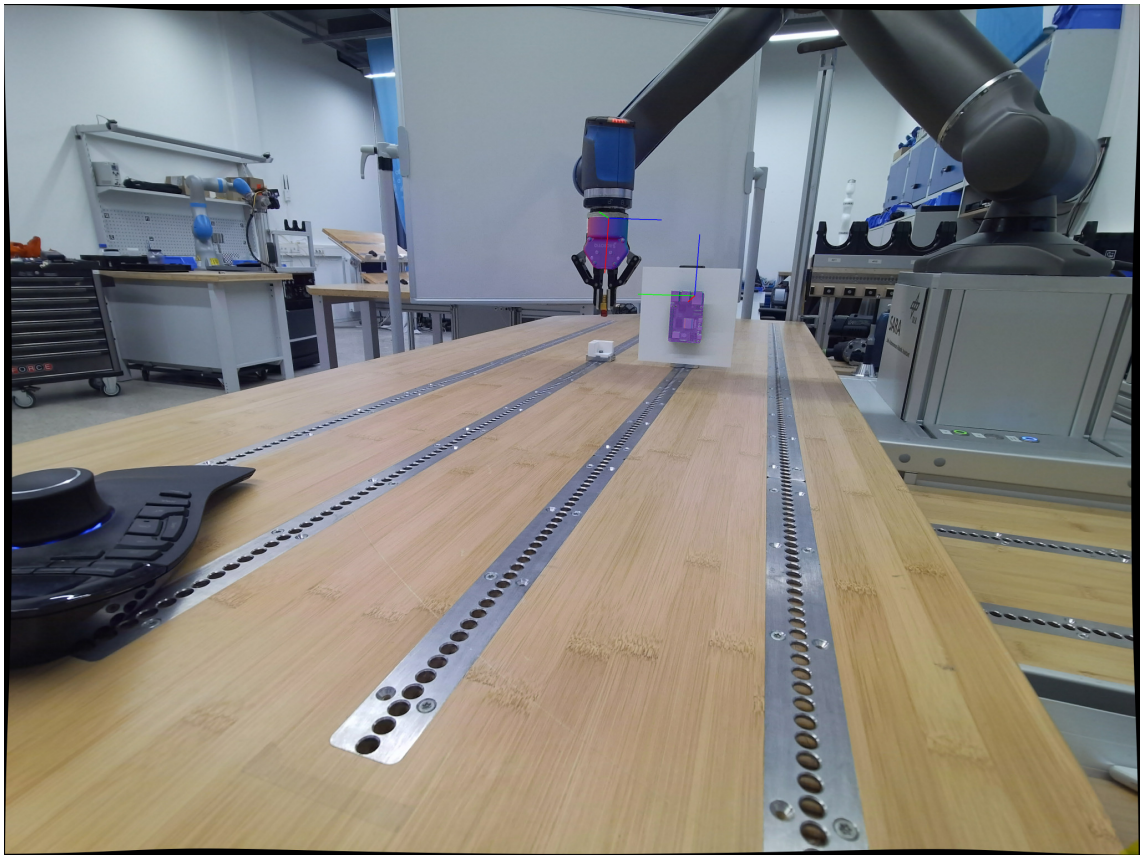


Figure 7.1: The rendered results of refining the poses for both the target device and the end-effector. This was used to estimate the robot's position independent of extrinsic camera calibration errors and kinematic inaccuracies. To improve robustness for estimating the end-effector pose, a whiteboard was placed in the background.

7.3 Real-World Experiments

The experiments conducted on the real robot demonstrate that EU-KGRL can compensate for positional inaccuracies during an in-contact insertion task. In contrast to KGRL, which struggles with positional offsets exceeding approximately 1 mm in the examined setup, EU-KGRL learns to reliably compensate for offsets of around 2 mm. This performance is achieved in under 40 episodes, which is comparable to the training time of the original KGRL algorithm in the setup examined in [8]. In the real-world setup examined in this thesis, training the robot for 50 episodes took approximately 30 minutes per run.

This can clearly be seen from the results of Experiment R1 presented in Chapter 6.3. KGRL consistently fails to solve the insertion task when the offset to the true target position is 2 mm. In contrast, EU-KGRL successfully learns to solve the task even under these conditions. Similarly, in Experiment R2, the success rate of EU-KGRL converges to 1, while KGRL does not improve. The Mean Positional Error of the target position estimates remained below 1 mm in every training run, but deviations exceeding 1 mm were frequently observed (see Table 6.17 and Appendix C). Simultaneously, EU-KGRL learns the secondary objective of minimizing contact forces with the environment, thereby reducing wear and tear on the target device, connector, and robot. However, it is unclear whether EU-KGRL generally optimizes the force interaction cost, or whether this effect is due to the robot registering fewer contact forces during successful episodes in which no collision occurs.

Experiment R1 and Experiment R3 show that, for small errors (below 1 mm), both KGRL and EU-KGRL perform comparably. In fact, they suggest that in those cases, no learning is required at all. In Experiment R1, KGRL is successful for positional target offsets of up to ± 1 mm throughout the training process, even in the first episode. In Experiment R3, where a consistent millimeter-accurate position estimate was used (see Chapter 6.2), both KGRL and EU-KGRL were successful from the beginning of the episode. An improvement in force interaction cost could also not be observed. This indicates that the underlying movement primitive, an LC-KMP without null-space modification, is already sufficient for solving the task.

However, despite these findings, the limited number of experiments does not allow for definitive conclusions about the exact target offsets at which KGRL and EU-KGRL do or do not work, or at what point learning becomes necessary. It is likely that this would also be influenced by numerous setup-specific factors, such as the quality of demonstrations, the chosen hyperparameters for KGRL and EU-KGRL, and the type of connector used for the insertion task. A comprehensive study would be necessary to fully assess the performance of KGRL and EU-KGRL across all these variables.

Possible limitations of the experimental setup need to be mentioned. The robot's impedance controller was set to be as stiff as possible, but it inevitably introduces minor variances due to limitations such as sensor noise and actuator dynamics. The Raspberry Pi was tightly fixed to the workbench, but tolerances in the hardware mounting and the flexibility of the connector itself contribute to the system's ability to passively compensate for small errors. As a result, the setup may mask differences between methods in the sub-millimeter range. Experiments on systems with tighter tolerances, such as the BNC connector setup used in prior work ([8]), could provide more conclusive insights.

In Experiment R2, where target positions obtained from the camera pose estimation pipeline were used, it could not be shown that policies learned at one target position generalize to other positions. First experiments deploying a learned policy to new, previously unseen target positions with variance from the pose estimation pipeline showed limited success. This problem is generally hard to solve due to variations in both the bias and orientation of the estimated pose distribution. The experiment setup requires the distribution to be corrected manually, such that the mean aligns with the ground truth. Therefore, the true target position needs to be known, which, in a real-world setup, would eliminate the need to compensate for pose estimation variance. Furthermore, since the exploration space is shaped by the covariance, changes in its

orientation (which would occur when viewing the target object from a different angle) directly affect the behavior of the learned policy. As a result, transferring policies across different configurations is non-trivial and requires further investigation.

7.4 Limitations of EU-KGRL

While EU-KGRL demonstrates clear advantages in certain scenarios, it also has limitations. The method relies on an accurate representation of uncertainty. In practical applications, it is difficult to capture all relevant sources of uncertainty, particularly when they arise from a complex perception pipeline. Without access to ground truth, correcting systematic errors is challenging, limiting the approach's applicability in real-world manufacturing settings.

Additionally, the use of static covariance does not fully exploit the strengths of kernel-based representations. The original motivation of KGRL lies in handling time-dependent uncertainty derived from demonstrations and generating a smooth exploration space based on the covariance obtained from the stochastic properties of the demonstrations. In contrast, the use of static external uncertainty may not justify the added complexity of EU-KGRL. Simpler approaches, such as residual policies that are activated only at specific phases of the task, may achieve similar results.

The nature of the insertion task introduces additional challenges. Manually increasing the variance near the target inevitably leads to higher interaction forces, which contradicts the safety-oriented design of KGRL. This trade-off between exploration and safety is particularly critical in contact-rich tasks.

Finally, it should be noted that, in its current form, EU-KGRL does not solve the problem of finding the connector itself. Instead, it only compensates for minor errors in target position estimates. The problem becomes increasingly difficult as the positional error grows. If the robot fails to make meaningful contact with the environment, the available sensory feedback is insufficient to guide learning. This could happen, for example, when the robot pushes the connector against the brim of the port in such a way that contact forces are measured only in the z-direction relative to the end-effector. In this case, even a learned policy does not have enough information to decide in which direction to correct the trajectory; the problem becomes partially observable. Learning complex exploration strategies is both challenging and highly sample-inefficient in real-world systems, especially for large exploration spaces. For testing purposes, Experiment R1 was repeated with an artificial centimeter-level offset. As expected, EU-KGRL was unable to solve the task of locating the connector within such a large workspace. The work in [13] shows that LfD approaches in combination with residual RL can solve such a task with offsets of up to $\pm$ 1 cm. However, this required a substantially higher training time of 500 episodes, with a success rate of around 70%, which may not be sufficient in an industrial setting.

8

Conclusion and Outlook

This work introduced External-Uncertainty KGRL (EU-KGRL), an extension of Kernel Guided Reinforcement Learning that incorporates externally-provided covariance into the LC-NS-KMP movement primitive framework. The objective was to enable learning in scenarios where the uncertainty is not fully captured by demonstrations.

The results show that EU-KGRL can successfully improve performance over KGRL when relevant external uncertainty is available and properly modeled. In both simulation and real-world experiments, the method demonstrated the ability to adapt trajectories in the presence of positional inaccuracies, including in contact-rich insertion tasks. This extension made it possible to apply the LfD-based KGRL framework to scenarios where it would otherwise fail.

To evaluate EU-KGRL in a real-world setting, where uncertainties may arise that are not accounted for in demonstrations, a camera-based pose estimation pipeline was implemented and analyzed. The pipeline achieved millimeter-level position accuracy and was integrated into an RL framework to learn to insert an Ethernet connector from user demonstrations. The real-world experiments highlight the importance of accurate uncertainty modeling. Even minor inaccuracies in the pose estimation pipeline limit the effectiveness of EU-KGRL and reveal fundamental challenges in integrating perception-driven uncertainty into learning-based control methods.

Overall, this work demonstrates that incorporating external uncertainty into KGRL is a promising direction, but also emphasizes the need for careful consideration of the underlying assumptions and system limitations.

Several directions for future work emerge from this thesis. While it was shown that EU-KGRL can be used to learn a connector-insertion task under target position uncertainty, the boundaries of its capabilities remain unclear. A comprehensive study that analyzes EU-KGRL under varying degrees of target pose uncertainty could clarify under which conditions it is necessary to modify the original KGRL formulation and when it may not be necessary to use reinforcement learning at all. This could also show in which scenarios EU-KGRL cannot correct the robot's trajectory based solely on contact force sensor readings. In this context, the effects of demonstration quality and hyperparameter choice on training performance should be analyzed for KGRL and EU-KGRL.

Additionally, different connector types should be explored for the insertion task. In this thesis, an Ethernet connector was selected due to its relatively high tolerance and

straightforward mating mechanism, making it a suitable choice for a proof-of-concept experiment. However, the strength of LfD approaches lies in providing an intuitive interface to program a robot to perform complex tasks. In [8], KGRL showed that introducing RL in a controlled way enables the use of LfD for fine mechanical in-contact tasks. For an Ethernet connector, the simple mating process and relatively high tolerances might not justify the use of RL or LfD in a real-world scenario, as simpler algorithms could solve the task. Choosing a connector with a complex mating process and very tight tolerances, as in the BNC experiment conducted in [8], would be a more suitable scenario for an algorithm like EU-KGRL. Additionally, it would provide a test setup with lower positional tolerance, allowing detailed assessment of EU-KGRL's performance for in-contact tasks.

Most importantly, future work should focus on incorporating different types of external variance into EU-KGRL and exploring its potential for an entirely new use case. Using target covariance obtained from a camera-based pose estimator is not ideal in a real-world setup, as it would require modeling every possible source of millimeter-level position estimate error or compensating for effects that cannot be modeled. At the same time, the fixed covariance from the position estimate error of a static object does not fully leverage the potential of the kernelized exploration-space representation that is used in EU-KGRL. A key extension would be the use of time- or state-dependent covariance. Instead of being limited to static uncertainty representations, future experiments could leverage the full expressive power of kernel-based methods to smoothly model uncertainty that evolves over time or depends on the robot's state. The original KGRL formulation shows that it can accurately model the time-dependent variance observed during demonstrations. Investigating the effect of external time- or state-dependent covariance could reveal the full potential of EU-KGRL. For example, [61] introduces a probabilistic kinematic model that captures configuration-dependent robot position errors. EU-KGRL could use this probabilistic model to learn to compensate for errors in the kinematic robot model, even for robot configurations that were not observed during demonstrations.

Bibliography

[1] Falco, J., Kimble, K., Van Wyk, K., Messina, E., Sun, Y., Shibata, M., Uemura, W., and Yokokohji, Y. Benchmarking Protocols for Evaluating Small Parts Robotic Assembly Systems. In: *IEEE Robotics and Automation Letters* 5:883–889, 2020. DOI: 10.1109/LRA.2020.2965869.

[2] Beck, L., Gebauer, D., Rauh, T., Dirr, J., and Daub, R. Deep Learning-Based Localization of Electrical Connector Sockets for Automated Mating. In: *Production Engineering* 19:187–194, 2025. DOI: 10.1007/s11740-024-01299-7.

[3] Iskandar, M., Ott, C., Eiberger, O., Keppler, M., Albu-Schäffer, A., and Dietrich, A. Joint-Level Control of the DLR Lightweight Robot SARA. In: *Proc. of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 8903–8910. DOI: 10.1109/IROS45743.2020.9340700.

[4] Iskandar, M., Eiberger, O., Albu-Schäffer, A., De Luca, A., and Dietrich, A. Collision Detection, Identification, and Localization on the DLR SARA Robot with Sensing Redundancy. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 3111–3117. DOI: 10.1109/ICRA48506.2021.9561677.

[5] Ravichandar, H., Polydoros, A., Chernova, S., and Billard, A. Recent Advances in Robot Learning from Demonstration. In: *Annual Review of Control, Robotics, and Autonomous Systems* 3:297–330, 2020. DOI: 10.1146/annurev-control-100819-063206.

[6] Schaal, S. Dynamic Movement Primitives - A Framework for Motor Control in Humans and Humanoid Robotics. In: *Adaptive Motion of Animals and Machines*. 2006, pp. 261–280. DOI: 10.1007/4-431-31381-8_23.

[7] Huang, Y., Rozo, L., Silvério, J. a., and Caldwell, D. G. Kernelized Movement Primitives. In: *The International Journal of Robotics Research* 38:833–852, 2019. DOI: 10.1177/0278364919846363.

[8] Padalkar, A., Stulp, F., Neumann, G., and Silvério, J. Towards Safe and Efficient Learning in the Wild: Guiding RL With Constrained Uncertainty-Aware Movement Primitives. In: *IEEE Robotics and Automation Letters* 10:6880–6887, 2025. DOI: 10.1109/LRA.2025.3566599.

[9] Huang, Y. and Caldwell, D. G. A Linearly Constrained Nonparametric Framework for Imitation Learning. In: *International Conference on Robotics and Automation*:4400–4406, 2020. DOI: 10.1109/ICRA40945.2020.9196821.

[10] Sundermeyer, M., Marton, Z.-C., Durner, M., Brucker, M., and Triebel, R. Implicit 3D Orientation Learning for 6D Object Detection from RGB Images. In: *The European Conference on Computer Vision (ECCV)*. 2018. DOI: 10.48550/arXiv.1902.01275.

[11] Elguea-Aguinaco, Í., Serrano-Muñoz, A., Chrysostomou, D., Inziarte-Hidalgo, I., Bøgh, S., and Arana-Arexolaleiba, N. A Review on Reinforcement Learning

for Contact-Rich Robotic Manipulation Tasks. In: *Robotics and Computer-Integrated Manufacturing* 81, 2023. DOI: https://doi.org/10.1016/j.rcim.2022.102517.

[12] Padalkar, A., Quere, G., Raffin, A., Silvério, J., and Stulp, F. Guiding Real-World Reinforcement Learning for In-Contact Manipulation Tasks with Shared Control Templates. In: *Autonomous Robots* 48, 2024. DOI: 10.1007/s10514-024-10164-6.

[13] Davchev, T., Luck, K. S., Burke, M., Meier, F., Schaal, S., and Ramamoorthy, S. Residual Learning From Demonstration: Adapting DMPs for Contact-Rich Manipulation. In: *IEEE Robotics and Automation Letters* 7(2):4488–4495, 2022. DOI: 10.1109/lra.2022.3150024.

[14] Ijspeert, A., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors. In: *Neural Computation* 25:328–373, 2013. DOI: 10.1162/NECO_a_00393.

[15] Gutierrez, N. and Beksi, W. *Movement Primitives in Robotics: A Comprehensive Survey*. 2025. DOI: 10.48550/arXiv.2601.02379.

[16] Lepetit, V., Moreno-Noguer, F., and Fua, P. EPnP: An Accurate O(n) Solution to the PnP Problem. In: *International Journal of Computer Vision* 81, 2009. DOI: 10.1007/s11263-008-0152-6.

[17] Ronneberger, O., Fischer, P., and Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28.

[18] Nibali, A., He, Z., Morgan, S., and Prendergast, L. *Numerical Coordinate Regression with Convolutional Neural Networks*. 2018. DOI: 10.48550/arXiv.1801.07372.

[19] Hodan, T., Haluza, P., Obdržálek, v., Matas, J., Lourakis, M., and Zabulis, X. T-LESS: An RGB-D Dataset for 6D Pose Estimation of Texture-Less Objects. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2017, pp. 880–888. DOI: 10.1109/WACV.2017.103.

[20] Hodan, T., Michel, F., Brachmann, E., Kehl, W., Buch, A. G., Kraft, D., Drost, B., Vidal, J., Ihrke, S., Zabulis, X., et al. BOP: Benchmark for 6D Object Pose Estimation. In: *Proceedings of the European Conference on Computer Vision (ECCV)*:19–35, 2018. DOI: 10.1007/978-3-030-01249-6_2.

[21] Ulmer, M., Durner, M., Sundermeyer, M., Stoiber, M., and Triebel, R. 6D Object Pose Estimation from Approximate 3D Models for Orbital Robotics. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2023, pp. 10749–10756. DOI: 10.1109/IROS55552.2023.10341511.

[22] Stoiber, M., Pfanne, M., Strobl, K. H., Triebel, R., and Albu-Schäffer, A. A Sparse Gaussian Approach to Region-Based 6DoF Object Tracking. In: *Computer Vision – ACCV 2020*. 2020, pp. 666–682. DOI: 10.1007/978-3-030-69532-3_40.

[23] Park, T., Märtens, M., Lecuyer, G., Izzo, D., and D'Amico, S. SPEED+: Next Generation Dataset for Spacecraft Pose Estimation across Domain Gap. In: 2021. DOI: 10.48550/arXiv.2110.03101.

[24] Paraschos, A., Daniel, C., Peters, J. R., and Neumann, G. Probabilistic Movement Primitives. In: *Advances in Neural Information Processing Systems*. Vol. 26. 2013, pp. 2616–2624. URL: https://proceedings.neurips.cc/paper_files/pape

r / 2013 / file / e53a0a2978c28872a4505bdb51db06dc - Paper . pdf (visited on 04/30/2026).

[25] Calinon, S., Guenter, F., and Billard, A. On Learning, Representing and Generalizing a Task in a Humanoid Robot. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37(2):286–298, 2007. DOI: 10.1109/TSMCB.2006.886952.

[26] Cohn, D., Ghahramani, Z., and Jordan, M. Active Learning with Statistical Models. In: *Journal of Artifical Intelligence Research* 4:705–712, 1996. URL: https://arxiv.org/abs/cs/9603104 (visited on 04/30/2026).

[27] Silvério, J. and Huang, Y. A Non-parametric Skill Representation with Soft Null Space Projectors for Fast Generalization. In: *IEEE International Conference on Robotics and Automation*:2988–2994, 2023. DOI: 10.1109/ICRA48891.2023.10161065.

[28] Deo, A. S. and Walker, I. D. Overview of Damped Least-Squares Methods for Inverse Kinematics of Robot Manipulators. In: *Journal of Intelligent and Robotic Systems* 14:43–68, I 1995. DOI: 10.1007/BF01254007.

[29] Schmidt, F. *Links and Nodes Documentation*. URL: https://links-and-nodes.readthedocs.io/en/latest/ (visited on 04/30/2026).

[30] *Links and Nodes GPLv3 Release*. URL: https://gitlab.com/links_and_nodes/links_and_nodes (visited on 04/30/2026).

[31] Strobl, K. H., Sepp, W., Fuchs, S., Paredes, C., Smisek, M., and Arbter, K. *DLR CalDe and DLR CalLab*. Institute of Robotics and Mechatronics, German Aerospace Center (DLR). URL: http://www.robotic.dlr.de/callab/ (visited on 04/30/2026).

[32] Forsyth, D. A. and Ponce, J. *Computer Vision, A Modern Approach*. 2003. ISBN: 0-12-379777-2.

[33] Weng, J., Cohen, P., and Herniou, M. Camera Calibration with Distortion Models and Accuracy Evaluation. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 14:965–980, 1992. DOI: 10.1109/34.159901.

[34] *Camera Calibration and 3D Reconstruction*. OpenCV. URL: https://docs.opencv.org/4.x/d9/d0c/group___calib3d.html (visited on 04/30/2026).

[35] *Camera Calibration*. OpenCV. URL: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html (visited on 04/30/2026).

[36] *SARA*. Institute of Robotics and Mechatronics, German Aerospace Center (DLR). URL: https://www.dlr.de/en/rm/research/robotic-systems/arms/sara (visited on 04/30/2026).

[37] *Azure Kinect DK Datasheet*. Microsoft. URL: https://news.microsoft.com/wp-content/uploads/prod/2019/06/Factsheet-Azure-Kinect-DK.pdf (visited on 04/30/2026).

[38] *Intel RealSense D400 Series Datasheet*. Intel. URL: https://cdrdv2-public.intel.com/841984/Intel-RealSense-D400-Series-Datasheet.pdf (visited on 04/30/2026).

[39] *Raspberry Pi 5*. Raspberry Pi Foundation. URL: https://www.raspberrypi.com/products/raspberry-pi-5/ (visited on 04/30/2026).

[40] *Product Information Portal: Raspberry Pi 5*. Raspberry Pi Foundation. URL: https://pip.raspberrypi.com/categories/892-raspberry-pi-5 (visited on 04/30/2026).

[41] *Azure Kinect Sensor SDK - Intrinsics*. Microsoft. URL: https://microsoft.github.io/Azure-Kinect-Sensor-SDK/master/struct_microsoft_1_1_azure_1_1_kinect_1_1_sensor_1_1_intrinsics.html (visited on 04/30/2026).

[42] Kirpes, C., Sly, D., and Hu, G. Value of the 3D Product Model Use in Assembly Processes: Process Planning, Design, and Shop Floor Execution. In: *Applied System Innovation* 4(2), 2021. DOI: 10.3390/asi4020039.

[43] Wang, C.-Y., Bochkovskiy, A., and Liao, H.-Y. M. *YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors*. 2022. DOI: 10.48550/arXiv.2207.02696.

[44] Denninger, M., Winkelbauer, D., Sundermeyer, M., Boerdijk, W., Knauer, M., Strobl, K. H., Humt, M., and Triebel, R. BlenderProc2: A Procedural Pipeline for Photorealistic Rendering. In: *Journal of Open Source Software* 8(82), 2023. DOI: 10.21105/joss.04901.

[45] *BlenderProc*. German Aerospace Center (DLR) - Institute of Robotics and Mechatronics (RM). URL: https://github.com/DLR-RM/BlenderProc (visited on 04/30/2026).

[46] Zelek, M. *Raspberry Pi 5*. GrabCAD. URL: https://grabcad.com/library/raspberry-pi-5-2 (visited on 04/30/2026).

[47] Stoiber, M. Closing the Loop: 3D Object Tracking for Advanced Robotic Manipulation. PhD thesis. Technische Universität München, 2023. URL: https://elib.dlr.de/203345/ (visited on 04/30/2026).

[48] Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. In: *Robotics: Science and System*, 2018. DOI: 10.15607/RSS.2018.XIV.019.

[49] Labbé, Y., Carpentier, J., Aubry, M., and Sivic, J. CosyPose: Consistent Multi-view Multi-object 6D Pose Estimation. In: *European Conference on Computer Vision*. 2020, pp. 574–591. DOI: 10.1007/978-3-030-58520-4_34.

[50] *3DObjectTracking*. German Aerospace Center (DLR) - Institute of Robotics and Mechatronics (RM). URL: https://github.com/DLR-RM/3DObjectTracking (visited on 04/30/2026).

[51] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-Baselines3: Reliable Reinforcement Learning Implementations. In: *Journal of Machine Learning Research* 22(268):1–8, 2021. URL: http://jmlr.org/papers/v22/20-1364.html (visited on 04/30/2026).

[52] Kuznetsov, A., Shvechikov, P., Grishin, A., and Vetrov, D. Controlling Overestimation Bias with Truncated Mixture of Continuous Distributional Quantile Critics. In: *Proceedings of the 37th International Conference on Machine Learning*. Vol. 119. Proceedings of Machine Learning Research. 2020, pp. 5556–5566. URL: https://proceedings.mlr.press/v119/kuznetsov20a.html (visited on 04/30/2026).

[53] *Stable-Baselines3 Documentation - TQC*. URL: https://sb3-contrib.readthedocs.io/en/master/modules/tqc.html (visited on 04/30/2026).

[54] Umeyama, S. Least-Squares Estimation of Transformation Parameters Between two Point Patterns. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13(4):376–380, 1991. DOI: 10.1109/34.88573.

[55] Kabsch, W. A Discussion of the Solution for the Best Rotation to Relate Two Sets of Vectors. In: *Acta Crystallographica Section A* 34(5):827–828, 1978. DOI: 10.1107/S0567739478001680.

[56] Kabsch, W. A Solution for the Best Rotation to Relate Two Sets of Vectors. In: *Acta Crystallographica Section A* 32(5):922–923, 1976. DOI: 10.1107/S0567739476001873.

[57] Siipola, T. *Aligning Point Patterns with Kabsch–Umeyama Algorithm*. 2021. URL: https://zpl.fi/aligning-point-patterns-with-kabsch-umeyama-algorithm/ (visited on 04/30/2026).

[58] Barros, D. Enhancing Probabilistic Imitation Learning with Robotic Perception for Self-Organising Robot Workstation. Bachelor's Thesis. Technical University of Munich, 2024. URL: https://elib.dlr.de/206091/ (visited on 04/30/2026).

[59] Flandin, G., Chaumette, F., and Marchand, E. Eye-in-hand / Eye-to-hand Cooperation for Visual Servoing. In: *Proceedings - IEEE International Conference on Robotics and Automation* 3:2741 –2746 vol.3, 2000. DOI: 10.1109/ROBOT.2000.846442.

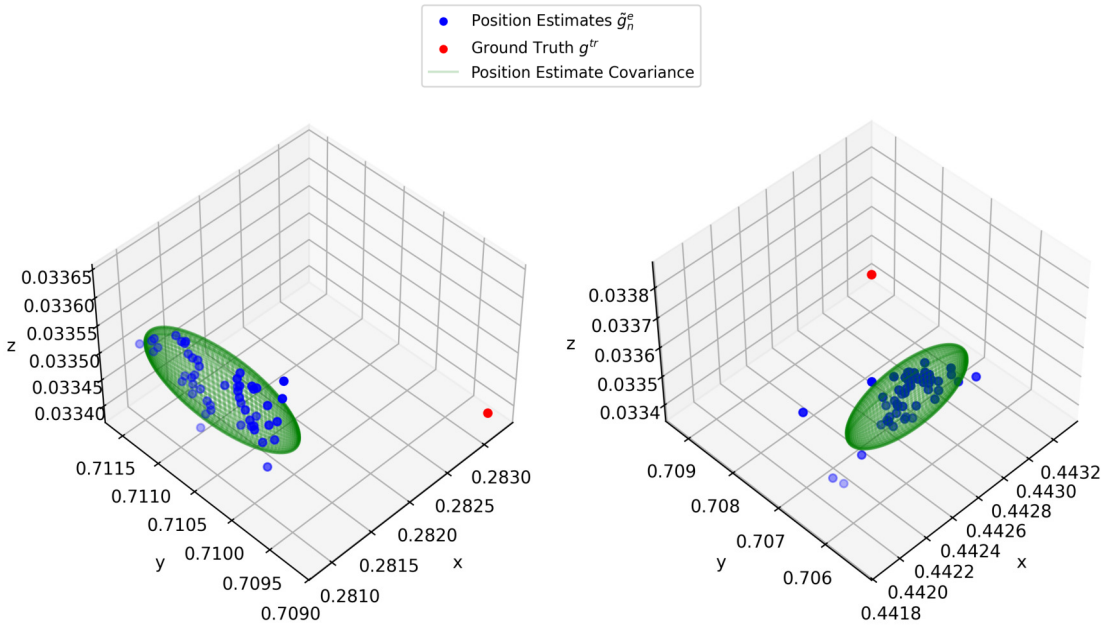
[60] *Technical Documentation - Archives*. Robotiq Inc. URL: https://robotiq.com/support#Archives (visited on 04/30/2026).

[61] Meyer, L., Strobl, K. H., and Triebel, R. The Probabilistic Robot Kinematics Model and its Application to Sensor Fusion. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 3263–3270. DOI: 10.1109/IROS47612.2022.9981399.

A

Recorded Positions used for the Pose Estimator Analysis (configured with Intel RealSense)

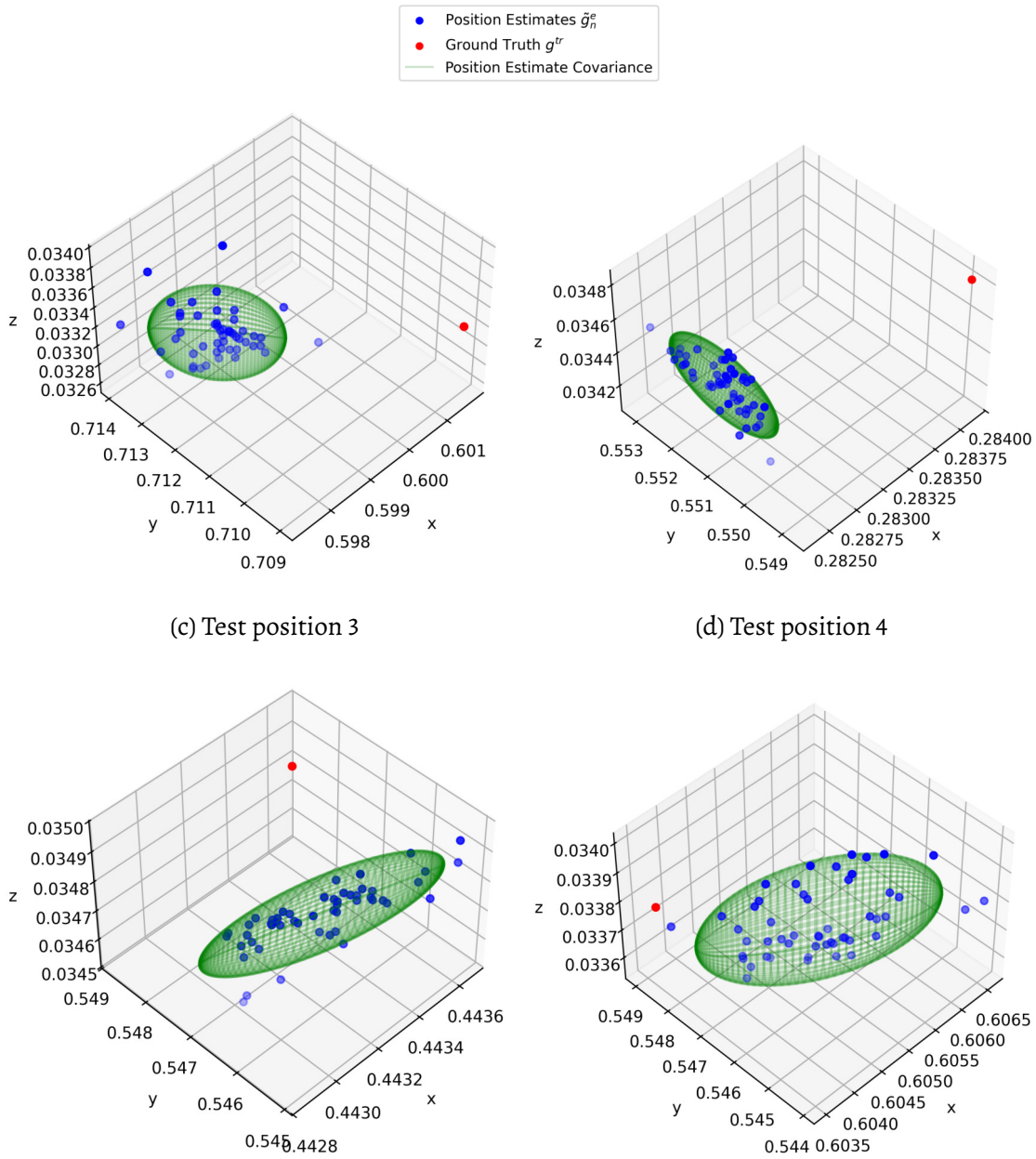
The position estimates $\tilde{g}_n^e$ used for the evaluation of the pose estimation pipeline that was done in Chapter 6.2 are presented here, together with the true test positions g^{tr}. The pipeline was configured with the Intel RealSense camera. The covariance that was constructed from the 50 position estimates is also shown. It is displayed using ellipsoids that represent the distribution around the mean of all estimates ($\tilde{\mu}^e$) with a standard deviation of 2. Note that, for better visualization, the axes are not uniformly scaled.



(a) Test position 1

(b) Test position 2

Figure A.1: Position estimates (for test positions 1 and 2) with corresponding ground truth and covariance for the analysis of the pose estimation pipeline, which was configured with the Intel RealSense camera. Positions are given in robot base frame and in meters.



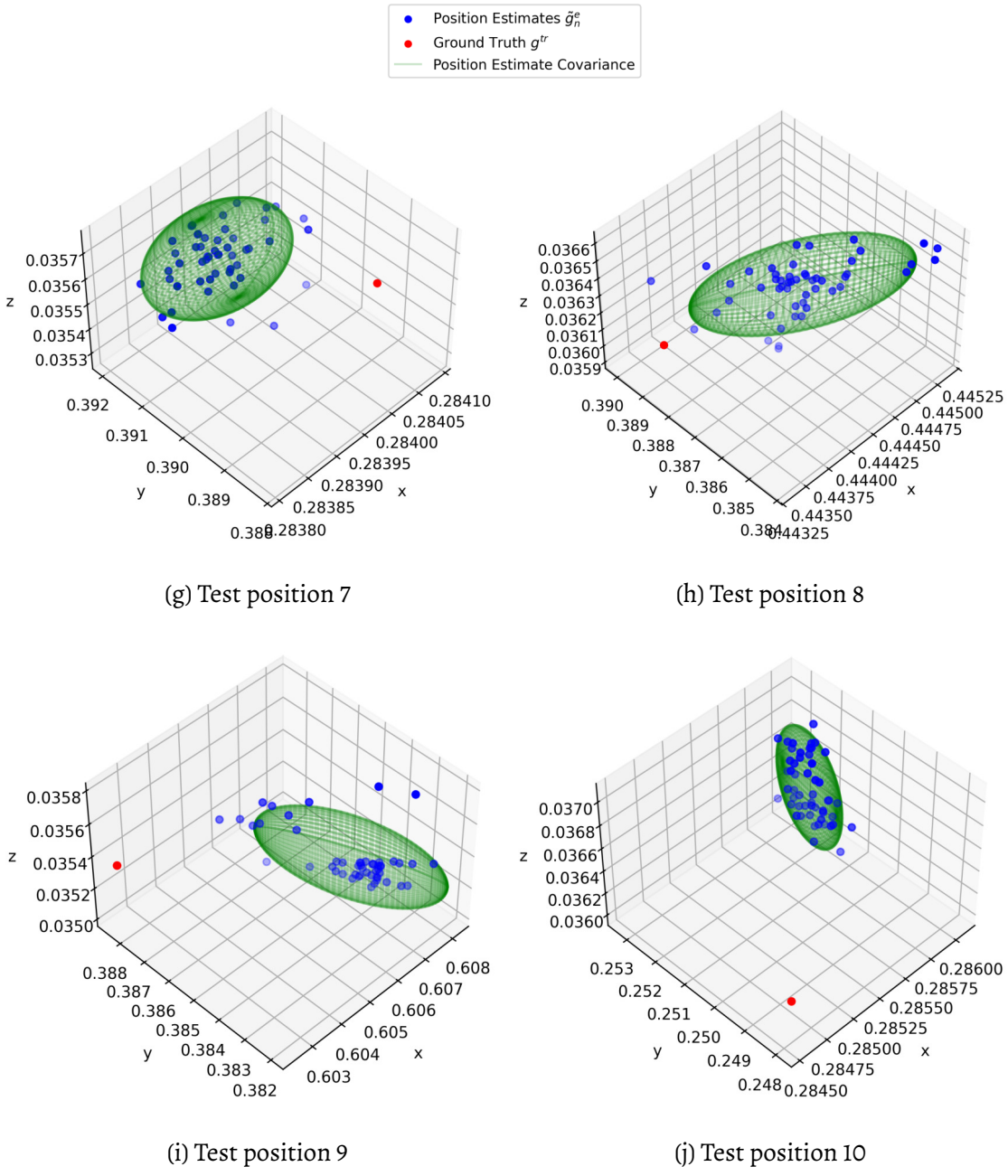
(c) Test position 3

(d) Test position 4

(e) Test position 5

(f) Test position 6

Figure A.1: Position estimates (for test positions 3 - 6) with corresponding ground truth and covariance for the analysis of the pose estimation pipeline, which was configured with the Intel RealSense camera. Positions are given in robot base frame and in meters.



(g) Test position 7

(h) Test position 8

(i) Test position 9

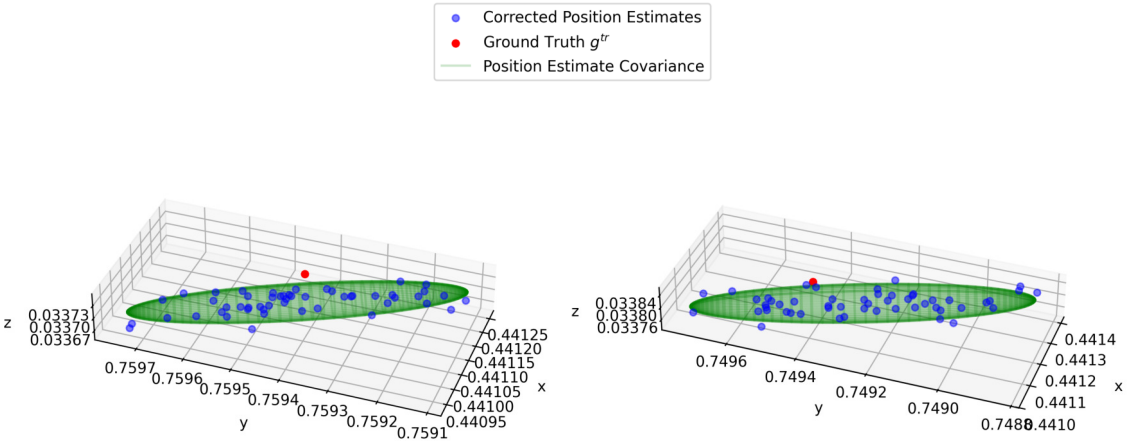
(j) Test position 10

Figure A.1: Position estimates (for test positions 7 - 10) with corresponding ground truth and covariance for the analysis of the pose estimation pipeline, which was configured with the Intel RealSense camera. Positions are given in robot base frame and in meters.

B

Recorded Positions used for the Pose Estimator Analysis (configured with Microsoft Azure Kinect)

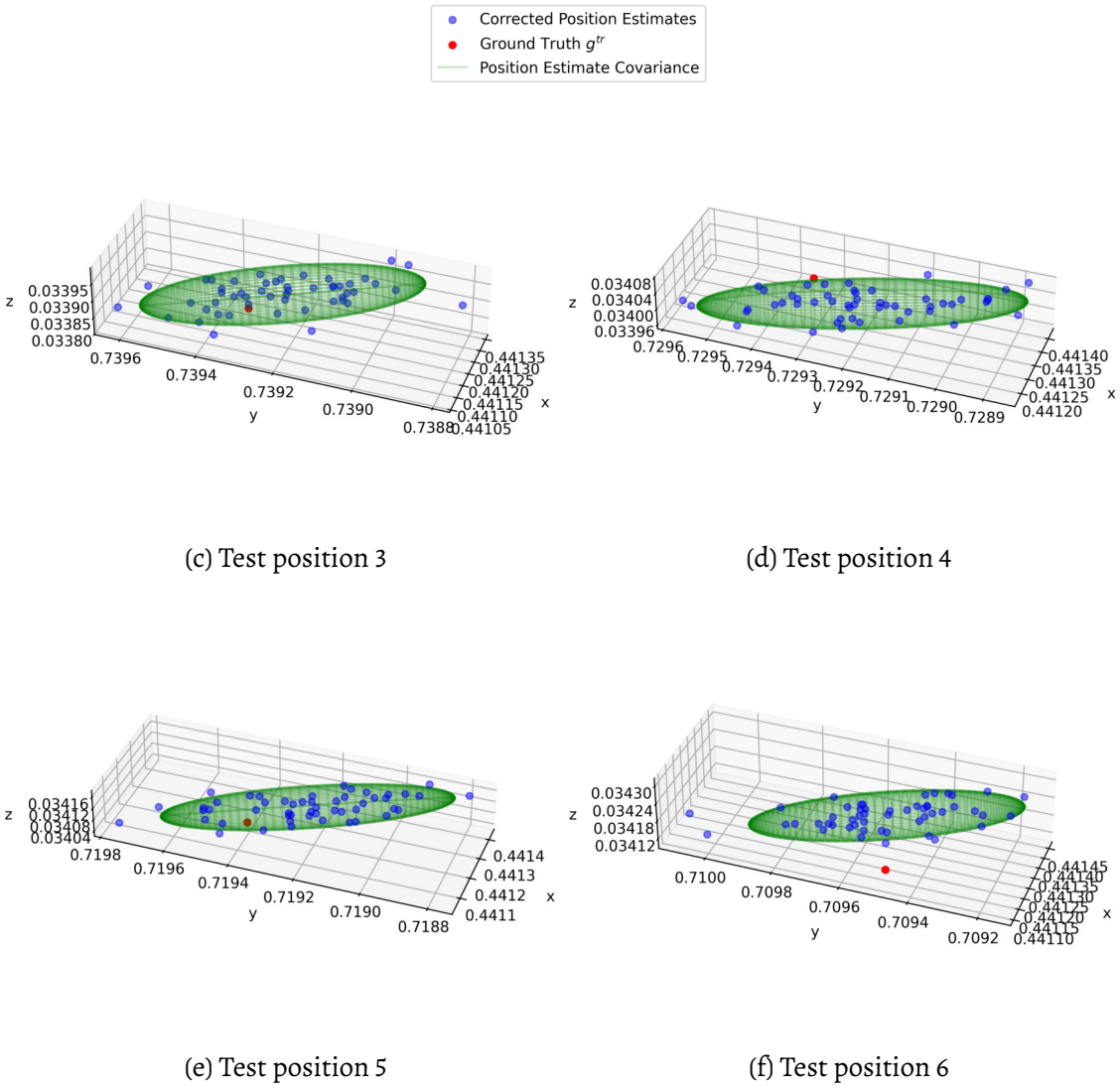
The corrected position estimates used for the evaluation of the pose estimation pipeline that was done in Chapter 6.2 are presented here, together with the true test positions g^{tr}. The pipeline was configured with the Microsoft Azure Kinect camera. The covariance that was constructed from the 50 position estimates is also shown. For displaying the covariance matrix, ellipsoids are used that represent the distribution around the mean of all corrected estimates with a standard deviation of 2. All positions are given in robot frame and in meters. Test positions 1 - 10 refer to the test positions that are placed closer together, with a distance of 1 cm between each other, as described in Chapter 6.2



(a) Test position 1

(b) Test position 2

Figure B.1: Corrected position estimates (for test positions 1 and 2) with corresponding ground truth, and covariance for the analysis of the pose estimation pipeline, which was configured with the Microsoft Azure Kinect camera. Positions are given in robot base frame and in meters.



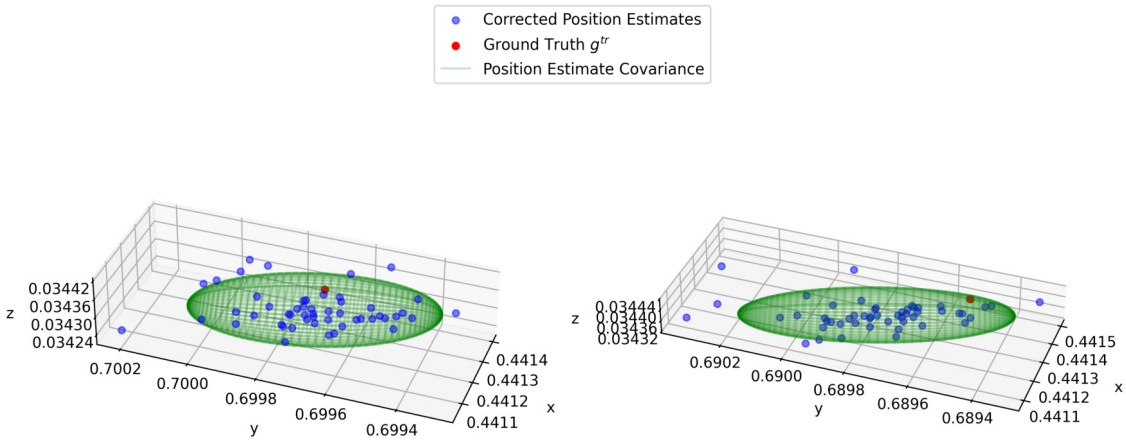
(c) Test position 3

(d) Test position 4

(e) Test position 5

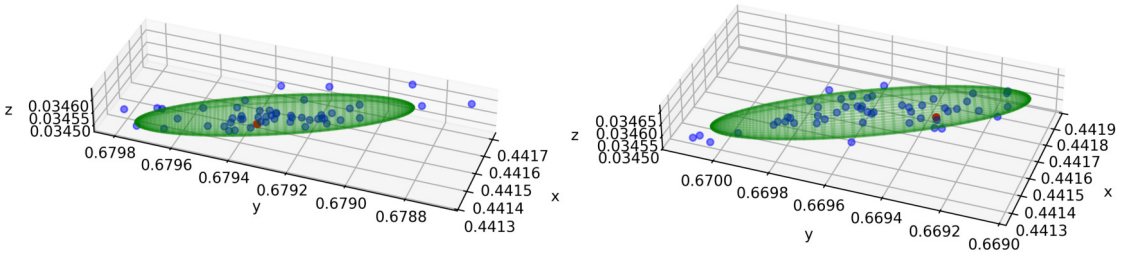
(f) Test position 6

Figure B.1: Corrected position estimates (for test positions 3 - 6) with corresponding ground truth, and covariance for the analysis of the pose estimation pipeline, which was configured with the Microsoft Azure Kinect camera. Positions are given in robot base frame and in meters.



(g) Test position 7

(h) Test position 8



(i) Test position 9

(j) Test position 10

Figure B.1: Corrected position estimates (for test positions 7 - 10) with corresponding ground truth, and covariance for the analysis of the pose estimation pipeline, which was configured with the Microsoft Azure Kinect camera. Positions are given in robot base frame and in meters.

C

Training Poses for Experiment R2

The positions of the corrected target position estimates $\tilde{g}$ used for training in Chapter 6.3 are presented here, together with the true target position g and the covariance obtained from the pose estimator. The covariance was constructed based on position estimates before training, while the position estimates $\tilde{g}$ were made during training. For displaying the covariance matrix, ellipsoids are used that represent the distribution around g with a standard deviation of 2. All positions are given in robot frame and in meters. For every training run, a new covariance matrix was generated by the pose estimator.

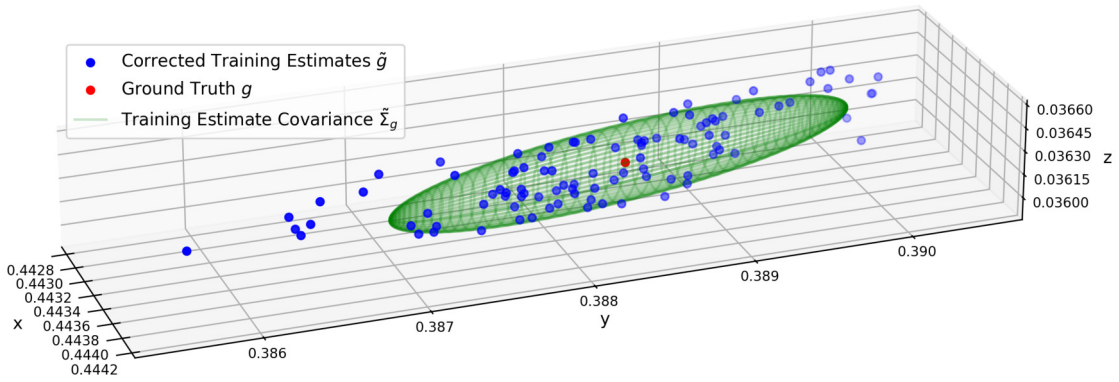


Figure C.1: Training poses for the robot experiment R2 observed during training run 1, together with the ground truth and covariance provided by the pose estimator. Positions are given in robot frame and in meters.

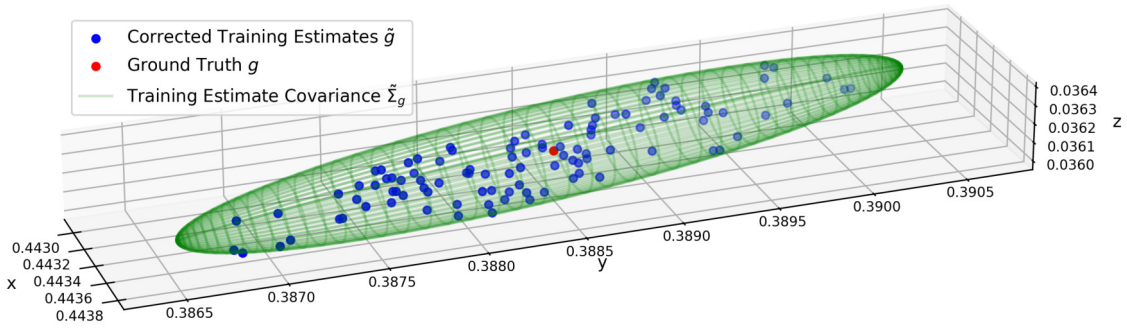


Figure C.2: Training poses for the robot experiment R2 observed during training run 2, together with the ground truth and covariance provided by the pose estimator. Positions are given in robot frame and in meters.

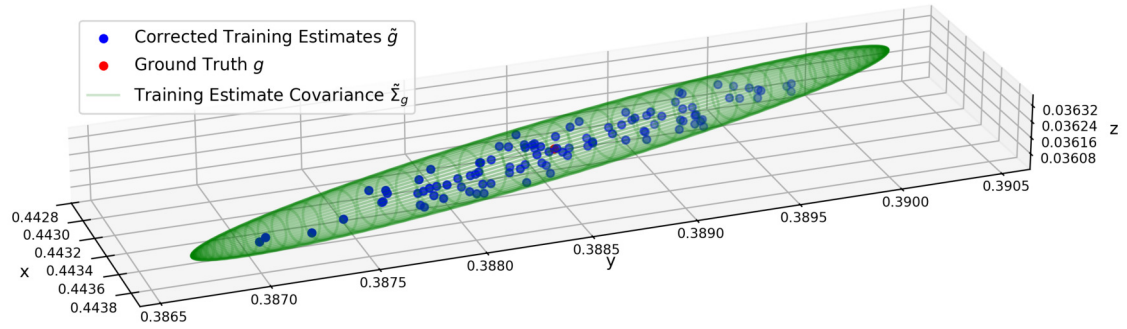


Figure C.3: Training poses for the robot experiment R2 observed during training run 3, together with the ground truth and covariance provided by the pose estimator. Positions are given in robot frame and in meters.

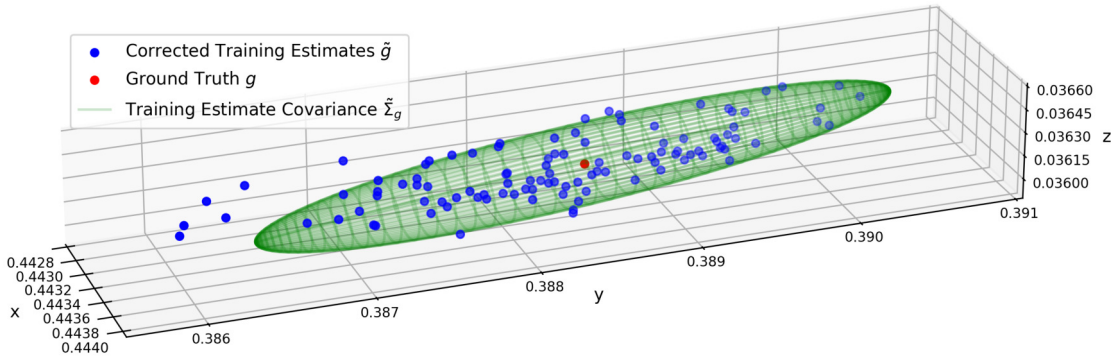


Figure C.4: Training poses for the robot experiment R2 observed during training run 4, together with the ground truth and covariance provided by the pose estimator. Positions are given in robot frame and in meters.

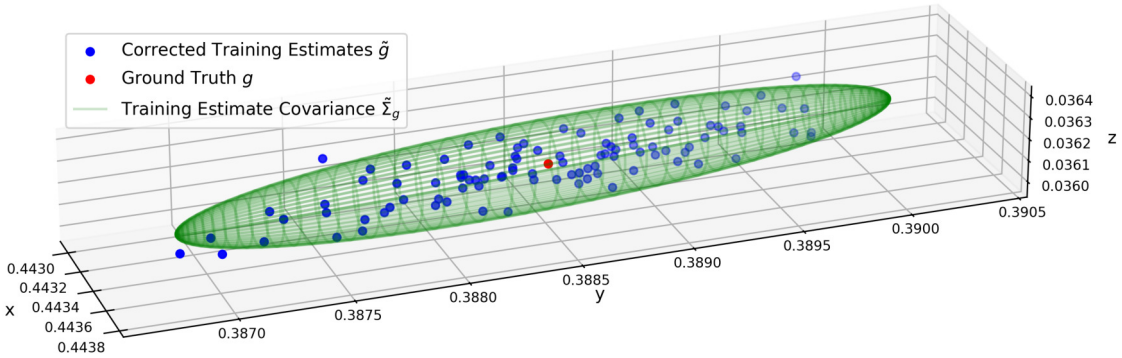


Figure C.5: Training poses for the robot experiment R2 observed during training run 5, together with the ground truth and covariance provided by the pose estimator. Positions are given in robot frame and in meters.