AUTONOMOUS GENERATION OF ROBOT BEHAVIOUR FOR EFFECTIVE TASK EXECUTION

Zvezdan Lončarević

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Prof. Dr. Andrej Gams, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia

Evaluation Board:
Prof. Dr. Tadej Petrič, Chair, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
Prof. Dr. Matjaž Mihelj, Member, Faculty of Electrical Engineering, Ljubljana, Slovenia
Prof. Dr. Mirko Raković, Member, Faculty of Technical Sciences, Novi Sad, Serbia

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Zvezdan Lončarević

AUTONOMOUS GENERATION OF ROBOT BEHAVIOUR FOR EFFECTIVE TASK EXECUTION

Doctoral Dissertation

AVTONOMNO OBLIKOVANJE OBNAŠANJA ROBOTA ZA UČINKOVITO IZVAJANJE NALOG

Doktorska disertacija

Supervisor: Prof. Dr. Andrej Gams

Ljubljana, Slovenia, October 2023

To my family and friends

Acknowledgments

The research for this thesis was performed at the Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute. First, I would like to thank my thesis supervisor Prof. Dr. Andrej Gams for his help and guidance during my studies. This research work was also done in collaboration with Dr. Mihael Simonič and Dr. Rok Pahič and I would like to thank them for their help. Together with Dr. Miha Deniša, Prof. Dr. Bojan Nemec and Prof. Dr. Leon Žlajpah I owe them special gratitude for their help, support and advice during the studies.

I want to thank the members of the evaluation board: Prof. Dr. Tadej Petrič, Prof. Dr. Mirko Raković and Prof. Dr. Matjaž Mihelj for reviewing the manuscript and evaluating the work of my thesis.

Finally, I would like to thank my family and friends for believing in me. Thank you for your unconditional support and love through all the stages of my life and studies.
This research has received funding from the:
- Program group P2-0076 Automation, robotics and biocybernetics supported by the Slovenian Research Agency,
- GOSTOP program C3330-16-529000, co-financed by Slovenia and EU under ERDF,
- EU's Horizon 2020 grant QU4LITY - Grant agreement No. 825030,
- Research grant Robot Textile and Fabric Inspection and Manipulation – RTFM (J2-4457) by the Slovenian Research Agency,
- Deutsche Akademische Austauschdienst (DAAD) - Grant No. 57588366.

Abstract

The pursuit of autonomous robot learning is aimed at enabling robots to seamlessly integrate into our daily lives, enhancing our efficiency and convenience. On the other hand, in the industrial sector, autonomous robot learning plays a vital role in keeping up with the rapidly evolving market demands. As products and processes change swiftly, robots equipped with autonomous learning capabilities can quickly adapt and optimize their performance, leading to increased productivity and competitiveness.

In this thesis, robot behaviour generation is considered at different levels of complexity. Initially, we address autonomous optimization methods, then progress to autonomous adaptation, and finally explore the potential of fully autonomous behaviour generation.

Installation or even just modification of robot-supported production and quality inspection is a tedious process that usually requires human expert engagement. The resulting parameters, e.g. robot velocities specified by an expert, are often subjective and suboptimal. That is why, in the first part of the thesis, we demonstrate an autonomous trajectory optimization method for the repetitive task. We propose a new approach for specifying visual inspection trajectories based on CAD models of workpieces to be inspected. The expert is required only to select the desired points on the inspection path along which the robot should move the camera. The rest of the approach is fully automatic.

In the second part of the thesis, we extend the autonomy of robot behaviour to adapting the whole industrial process. Robotic visual inspection is based on manually pre-defined postures between the camera and the object, and often a robot has to move the in-hand camera around of the object. The path of the robot is typically always the same, determined in advance. But, in order to check only the selected subset out of all possible aspects of the product, we need to generate all possible transitions between all possible aspects. This is a slow and tedious process. Therefore, we propose a method able to use neural networks in order to classify parts only according to possible defects. The classification results allow for the exclusion of certain viewpoints, effectively saving time on the production line. The emphasis lies on the adaptive nature of this industrial task, showcasing the ability to accelerate processes and improve overall efficiency in industrial settings.

Trajectory generation for dynamic tasks, which involve complex problem-solving and require more than simple motion repetition, is explored in the third part of the thesis. Due to the time-consuming nature of robot learning, simulations are often employed to accelerate the process. While complete skill learning in simulations is extensively researched and falls outside the scope of this part, our focus lies on effectively transferring knowledge from simulation to the real-world. Initially, a deep neural network encodes the skill in one domain, which is later adapted for a target domain by partial retraining using the real data. Our proposed approach combines backpropagation and reinforcement learning for retraining, demonstrating significant speed improvements compared to individual methods and a tenfold acceleration compared to learning from scratch. As an example for this approach, we apply it to the challenging task of robot throwing, which involves dynamic elements and is not reliant only on the final position of the robot motion.

Povzetek

Cilj prizadevanj za avtonomno učenje robotov je omogočiti robotom, da se nemoteno vključijo v naše vsakdanje življenje ter povečajo našo učinkovitost in udobje. Po drugi strani pa ima v industrijskem sektorju avtonomno učenje robotov ključno vlogo pri sledenju hitro razvijajočim se zahtevam trga. Ker se izdelki in procesi hitro spreminjajo, se lahko roboti, opremljeni s sposobnostmi avtonomnega učenja, hitro prilagodijo in optimizirajo svoje delovanje, kar vodi k večji produktivnosti in konkurenčnosti.

V tej disertaciji je generiranje obnašanja robotov obravnavano na različnih ravneh kompleksnosti. Na začetku naslovimo metode avtonomne optimizacije, nato preidemo na avtonomno prilagajanje in nazadnje raziskujemo možnosti popolnoma avtonomnega generiranja obnašanja.

Namestitev ali celo samo modifikacija robotsko podprte proizvodnje in kontrole kakovosti je zahteven proces, ki običajno zahteva sodelovanje človeškega strokovnjaka. Dobljeni parametri, npr. hitrosti robota, ki jih določi strokovnjak, so pogosto subjektivni in neoptimalni. Zato v prvem delu doktorske naloge prikažemo metodo avtonomne optimizacije trajektorije za ponavljajočo se nalogo. Predlagamo nov pristop za generiranje trajektorij vizualnega pregleda obdelovancev na podlagi CAD-modelov. Strokovnjak mora izbrati le želene točke na poti pregleda, po katerih naj robot premika kamero. Preostali del pristopa je popolnoma samodejen.

V drugem delu disertacije razširimo avtonomijo obnašanja robota na prilagajanje celotnega industrijskega procesa. Robotsko vizualno pregledovanje temelji na vnaprej določenih položajih med kamero in predmetom, pogosto pa mora robot premikati kamero v roki okoli predmeta. Pot robota je običajno vedno enaka, določena vnaprej. Da pa bi lahko preverili le izbrano podmnožico od vseh možnih vidikov izdelka, moramo ustvariti vse možne prehode med vsemi možnimi vidiki. To je počasen in zamuden postopek. Zato predlagamo metodo, ki lahko uporabi nevronsko mrežo za razvrščanje delov samo glede na možne napake. Rezultati klasifikacije omogočajo izključitev nekaterih vidikov, kar učinkovito prihrani čas na proizvodni liniji. Poudarek je na prilagodljivi naravi te industrijske naloge, ki prikazuje sposobnost pospeševanja procesov in izboljšanja splošne učinkovitosti v industrijskih okoljih.

V tretjem delu disertacije je raziskano ustvarjanje poti za dinamične naloge, ki vključujejo kompleksno reševanje problemov in zahtevajo več kot le preprosto ponavljanje gibanja. Zaradi dolgotrajne narave učenja robotov se za pospešitev procesa pogosto uporabljajo simulacije. Medtem ko je popolno učenje veščin v simulacijah obsežno raziskano in ne sodi v okvir tega dela, se osredotočamo na učinkovit prenos znanja iz simulacij v realni svet. Na začetku globoka nevronska mreža kodira spretnost v eni domeni, ki jo pozneje z delnim prekvalificiranjem z uporabo realnih podatkov prilagodi za ciljno domeno. Naš predlagani pristop združuje povratno razširjanje in učenje z ojačitvijo za prekvalifikacijo, pri čemer izkazuje znatne izboljšave hitrosti v primerjavi s posameznimi metodami in desetkratni pospešek v primerjavi z učenjem od začetka. Kot primer tega pristopa ga uporabimo za zahtevno nalogo robotskega metanja, ki vključuje dinamične elemente in ni odvisna le od končnega položaja gibanja robota.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

ACO	...	Ant Colony Optimization
AE	...	Autoencoder
AI	...	Artificial Intelligence
CAD	...	Computer-Aided Design
CAM	...	Computer-Aided Manufacturing
CD	...	Cell Decomposition
CDMP	...	Cartesian Dynamic Movement Primitives
CHOMP	...	Covariant Hamiltonian Optimization for Motion Planning
CMA−ES	...	Covariance Matrix Adaptation Evolution Strategy
CNC	...	Computerized Numerical Control
DMP	...	Dynamic Movement Primitives
DoF	...	Degree of Freedom
GA	...	Genetic Algorithms
ILC	...	Iterative Learning Control
LED	...	Light Emitting Diode
MC	...	Monte Carlo architecture for encoder-decoder
MLP	...	Multilayer-Perceptron
NN	...	Neural Network
NOK	...	not OK
OMPL	...	Open Motion Planning Library
PB	...	Parameter-Based architecture for encoder-decoder
PC	...	Personal Computer
PCA	...	Principal Component Analysis
PI^2	...	Policy Improvement with Path Integrals
PoWER	...	Policy Learning by Weighting Exploration with the Returns
PRM	...	Probabilistic Road-Map
ProDMP	...	Probabilistic Dynamic Movement Primitives
PSO	...	Particle Swarm Optimization
RG	...	Reality Gap
RL	...	Reinforcement Learning
RNN	...	Recurrent Neural Networks
ROS	...	Robot Operating System
RRT	...	Rapidly-Exploring Random Trees
RWPL	...	Reward-Weighted Policy Learning
SBP	...	Sampling-Based Methods for Planning
SBPL	...	Search-Based Planning Library
STOMP	...	Stochastic Trajectory Optimization for Motion Planning
TL	...	Transfer Learning
T−RRT	...	Transition-based Rapidly-Exploring Random Trees

Symbols

$\hat{b}_r$	...	bias of neural network
c_c	...	cross section of the manipulability measure and smallest normalized distance for point c
c_i	...	center of i-th radial basis function
$\mathbf{ctx}$	...	context point used as input to encoder-decoder network
c	...	number of clusters used in classification of the defect
$\boldsymbol{D}_p$	...	CDMP scaling matrix for position
$\boldsymbol{D}_o$	...	CDMP scaling matrix for orientation
e_n	...	ILC error in repetition n
d_f	...	distance between camera and object
$\mathbf{d}$	...	points distributed along the planned path
f	...	focal distance of the lens
$\boldsymbol{f}_p$	...	CDMP position forcing term
$\boldsymbol{f}_o$	...	CDMP orientation forcing term
$\boldsymbol{g}^p$	...	CDMP goal position
$\boldsymbol{g}^o$	...	CDMP goal orientation
H	...	height of the image
h_i	...	width of i-th radial basis function
$I(x, y)$	...	the intensity values at pixel (x, y)
K	...	number of trajectory segments
K_j	...	number of points in j-th segment
L	...	length of the generated trajectory
L_j	...	overall arc length of the j-th segment
$\boldsymbol{L}_{w_g}$	...	Cholesky decomposition of the covariance matrix of ProDMP parameters distribution in encoder-decoder architecture
M	...	number of weights used to define velocity profile
N	...	number of weights used to define the shape of the trajectory
$\boldsymbol{n}$	...	normals to the workpiece surface
$\boldsymbol{n}_{j,k}$	...	unit surface normals at points $\tilde{\boldsymbol{p}}_{j,k}$
$\mathbf{P}$	...	posterior probabilities of latent space parameters belonging to each of c GMM clusters
P	...	number of image samples taken in iteration of ILC learning
$\tilde{\boldsymbol{p}}_{j,k}$	...	positions of the extracted path at k-th point of j-th trajectory segment
$\boldsymbol{p}_{j,k}$	...	positions of time parametrized trajectory at k-th point of j-th trajectory segment
$\dot{\boldsymbol{p}}_{j,k}$	...	linear velocities of time parametrized trajectory at k-th point of j-th trajectory segment
$\ddot{\boldsymbol{p}}_{j,k}$	...	linear accelerations of time parametrized trajectory at k-th point of j-th trajectory segment
$\dot{p}_x$	...	manually calculated camera speed to image motion
Q	...	low-pass filter for smoothening ILC signal

$\tilde{\boldsymbol{q}}_{j,k}$ ... orientations in quaternions of the extracted path at k-th point of j-th trajectory segment

$\boldsymbol{q}_{j,k}$ ... orientations in quaternions of time parametrized trajectory at k-th point of j-th trajectory segment

$\boldsymbol{q}_w$ ... orientation in quaternions of the workpiece in the robot coordinate system

R ... reward in RWPL

$\boldsymbol{R}$ ... 3-D rotational matrix specifying the orientation of the camera

$r_{j,k}$ ... weighted sum of arc and angle length along the path at k-th point of j-th trajectory segment

$r_{h,n,p}$ ... intermediate reward computed at the p-th image captured in the n-th trial

$r_{v,n,p}$ ... intermediate reward computed at the p-th image captured in the n-th trial

$\hat{r}_z$ ... reward for one target

S ... trajectory smoothness

$s_{j,k}$ ... arc length along the path at k-th point of j-th trajectory segment

$\boldsymbol{T}$ ... desired landing spot coordinates

$\boldsymbol{T_t}$ ... evaluation landing spot coordinates

$\boldsymbol{T_r}$ ... reward landing spot coordinates

T_e ... the duration of the motion execution

T_j ... duration of inspection movement for j-th segment

$\boldsymbol{T'}$ ... real landing spot coordinates

$\boldsymbol{t}$ ... tangents of the path

$\boldsymbol{t}_w$ ... position of the workpiece in the robot coordinate system

$t_{j,k}$ ... time at which k-th point of j-th trajectory segment is reached

t_p ... time at which image sample is taken

t_e ... camera exposure time

u_n ... ILC input signal in repetition n

u_x ... pixel size

V ... width of the image

$\mathbf{V}_i$ ... inspection viewpoints associated with a GMM cluster

$\boldsymbol{V}^l$ ... latent space values of CDMP parameters

$\mathbf{W}$ ... trajectory parameters

w_i^ν ... CDMP weights defining velocity profile

$\boldsymbol{w}_i^p$ ... CDMP weights defining the shape of position part of the trajectory

$\boldsymbol{w}_i^o$ ... CDMP weights defining the shape of orientation part of the trajectory

x ... CDMP phase

y ... image quality measure

$\boldsymbol{y}$ ... ground truth trajectory used to train encoder-decoder NN

$\boldsymbol{y}_b$ ... robot EE position used as boundary condition for integrating ProDMP trajectory

$\dot{\boldsymbol{y}}_b$ ... robot EE velocity used as boundary condition for integrating ProDMP trajectory

y_d ... desired image quality measure

y_n ... ILC output signal in repetition n

Z ... number of targets used in one iteration of learning

$\boldsymbol{z}$ ... scaled linear velocity

α ... constant used for time parametrization

α_x ... CDMP parameter

β ... constant used for time parametrization

γ ... constant used to stop ILC before slowing down too much

δ ... angle between consecutive planned trajectory segments

ε ... allowed deviation from the desired image quality measure

$\boldsymbol{\varepsilon}_n$ ... zero-mean Gaussian noise used for exploration in RWPL

ζ_j ... overall angle length of the j-th segment
$\boldsymbol{\eta}$ ... scaled angular velocity
κ ... user-selected constant for terminal reward
λ ... ILC gain for defining the update step
$\boldsymbol{\mu}_A$ ... mean of the prior distribution used in encoder-decoder architecture
μ_c ... manipulability measure for point c
$\boldsymbol{\mu}_r$ ... mean of the latent space observation distribution used in encoder-decoder architecture
$\boldsymbol{\mu}_\tau$ ... mean of the predicted trajectory distribution using encoder-decoder architecture
$\hat{\mu}_c$ ... normalized manipulability measure for point c
$\boldsymbol{\mu}_{w_g}$ ... mean of the ProDMP parameters distribution in encoder-decoder architecture
$\boldsymbol{\mu}_z$ ... mean of the latent space distribution in encoder-decoder architecture
$\nu(x)$ ... speed scaling function
$\xi_{j,k}$ ... angle length along the path at k-th point of j-th trajectory segment
ρ_c ... smallest normalized distance to the joint limit for point c
τ ... CDMP constant defining the initial duration of motion
$\boldsymbol{\tau}$ ... encoder-decoder network predicted trajectory
τ_{slow} ... initial slow trajectory execution time
$\boldsymbol{\Sigma}_\tau$ ... covariance of the predicted trajectory distribution using encoder-decoder architecture
$\boldsymbol{\sigma}_A^2$ ... variance of the prior distribution used in encoder-decoder architecture
$\boldsymbol{\sigma}_r^2$ ... variance of the latent space observation distribution used in encoder-decoder architecture
$\boldsymbol{\sigma}_{\boldsymbol{\tau}}^{\mathbf{2}}$ ... variance of the predicted trajectory distribution using encoder-decoder architecture
$\boldsymbol{\sigma}_z^2$ ... variance of the latent space distribution in encoder-decoder architecture
$\boldsymbol{\theta}$ ... encoder-decoder weights and biases
$\boldsymbol{\Theta}_n$ ... RWPL set of policy parameters executed until n-th iteration
$\boldsymbol{\theta}_n^*$ ... parameters of n-th iteration with added exploration noise
$\boldsymbol{\theta}_n$ ... RWPL policy parameters in n-th iteration
θ^{AE} ... latent space projection of the production parameters
ϕ_h ... squared magnitude of the horizontal gradient
ϕ_v ... squared magnitude of the vertical gradient
ϕ ... focus measure
ϕ_r ... relative focus measure
$\phi_{h,n,p}$ ... horizontal image quality measures computed at the p-th image captured in the n-th trial
$\phi_{v,n,p}$ ... vertical image quality measures computed at the p-th image captured in the n-th trial
$\phi_{h,d,p}$ ... horizontal reference image quality measure interpolated to the phase of p-th image captured in the n-th trial
$\phi_{v,d,p}$ ... vertical reference image quality measure interpolated to the phase of p-th image captured in the n-th trial
$\phi_{h,\text{ILC},p}$ ... horizontal aligned and interpolated image quality measures obtained when executing the inspection trajectory generated from the parameters computed by ILC
$\phi_{v,\text{ILC},p}$ ... vertical aligned and interpolated image quality measures obtained when executing the inspection trajectory generated from the parameters computed by ILC
Ψ_i ... i-th radial basis function used in CDMP
$\boldsymbol{\omega}_{j,k}$ ... angular velocities of time parametrized trajectory at k-th point of j-th trajectory segment

$\dot{\boldsymbol{\omega}}_{j,k}$ ... angular accelerations of time parametrized trajectory at k-th point of j-th trajectory segment
$\hat{\omega}_r$ ... weight of neural network
$\mathcal{D}_{\text{so}}$ ... source environment dataset
$\hat{\mathcal{D}}_{\text{so}}$ ... dataset connecting targets and latent space values obtained in source environment
$\mathcal{D}_{\text{ta}}$ ... dataset connecting targets and trajectory parameters obtained in target environment
N_{BP} ... neural network retrained using BP
N_{RL} ... neural network retrained using RL
$N_{\text{RL\&BP}}$ ... neural network retrained using a combination of BP and RL
N_{so} ... neural network trained using source environment data
N'_{so} ... neural network trained using source environment data with only one layer unlocked
N'_{ta} ... neural network retrained using target environment data with only one layer unlocked
$\langle\cdot\rangle_{w(\boldsymbol{\pi}_k)}$ ... importance sampling

Chapter 1

Introduction

Rising demand for a huge spectre of products with different features and characteristics has led to the increased demands for faster and better adaptations of production lines between highly competitive companies around the world. Such high-mix low-volume production strategy caused big changes in hardware and software design. Hardware needs to be capable of quickly adapting to the new series of products, and therefore the whole spectre of adaptive, reconfigurable workcells has emerged [1]–[3]. Newly developed hardware also required fast adaptation of the software, which caused the field of robotic learning to become an inevitable tool in nowadays engineering. It provides tools for the design of sophisticated and hard-to-engineer robot tasks without the need for manual programming [4]. These tools can be used for optimization, prediction and planning, as well as learning new skills entirely from scratch. It is expected that the number and variety of robot applications will increase in the factories of the future [5].

The optimization of robot trajectories to achieve faster execution times is a common challenge in the field of industrial robotics. Minimum-time trajectory planning has been extensively researched and developed, utilizing both offline and online methodologies [6]. Approaches for optimizing the trajectories with respect to energy [7], actuation effort [8] or both [9] have already been developed, but they are model-based and cannot be applied when the relationship between the optimization criterion and optimization variables is not known, as in the case of vision. Alongside the other sensory systems that production lines heavily rely on, vision is becoming a key driving force behind manufacturing processes in modern factories [10], thanks to its ability to detect the position and identity of different parts. Consequently, image quality is recognized as one of the main essential aspects of visual inspection systems [11]. A wide array of visual sensors such as structured light, time of flight, line scan cameras, stereo vision as well as monocular vision with standard area scan or RGB cameras are already employed for visual inspection and quality control processes [12], [13]. The application of robot-assisted visual quality inspection has a whole variety of possible applications, but there are two main approaches for conducting such inspections: a robot can examine a workpiece from a limited number of specific angles or capture a continuous stream of visual data. Additionally, the robot may either hold a camera and adjust its position in relation to the workpiece or, alternatively, maintain a fixed camera position while manipulating the workpiece being inspected. In the first part of the thesis, we focus on the example where the continuous stream of data is being recorded and a robot is moving the camera around a stationary object - Figure 1.1. The main problem of this approach is that the movement of the robot needs to be optimized so that we acquire the whole stream of data in the shortest possible time without affecting the sharpness of the image. For achieving a sharp image in our case, the robot always needs to be at a constant distance from the object surface (note that industrial cameras usually

have fixed focal length) and its velocity should not be too high in order to prevent motion blur. The process of setting up such a system is a tedious job for the operator and might at the end result in trajectories that are sub-optimal. A typical example of such a quality inspection process is the inspection of weld seams [14]. The determination of the correct distance from an object is used as described in [15], while our focus is on the algorithm capable of determining the optimal velocity.

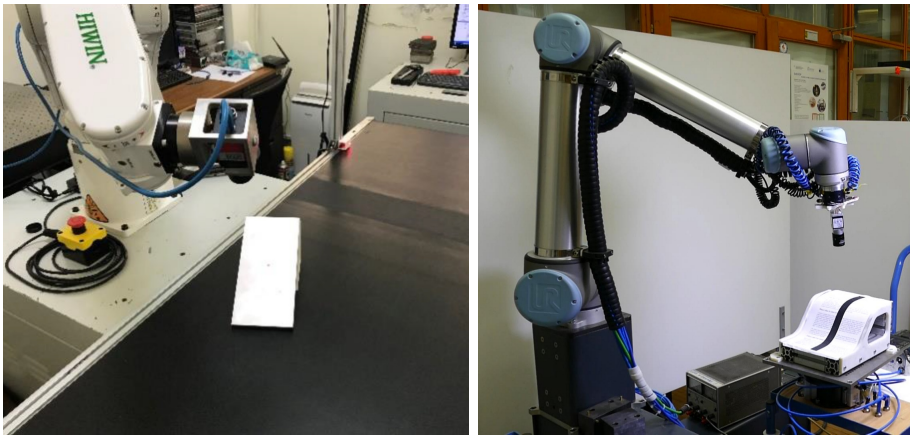


Figure 1.1: Optimization of a robot inspection process with continuous stream of visual data being recorded using an in-hand industrial camera. Left picture reprinted from [16], used under Creative Commons CC-BY license.

Optimization of one trajectory is a common and widely spread task in robotics and assumes that the path is already predefined. Therefore, it can be applied only to repetitive industrial processes with a hard-defined pipeline. The more complex problem arises in situations when the robot is holding a camera and needs to inspect the workpiece from many different viewpoints and has to place it so that there are no obstacles in the line of sight [17] as shown in Figure 1.2. These setups have a lot of trajectories that need to be generated even when the number of discrete viewpoints is fixed. Such a process usually takes a lot of time, but the pace of the production is often relatively high, thus leaving only a little time for the robot to move the camera around the object. Fast and accurate motions need to be implemented in such cases, but even that might not be enough. That is why extensions to the manufacturing lines that include buffers, where the products wait for inspection, can be implemented, or parallel checking with several cameras on several robots can be employed. Introducing all of these might increase production costs and therefore a trade-off is struck between the possible number of defects not being detected and the effort put into quality inspection [18]. To avoid checking all the aspects of the product, we propose using AI approaches to predict at which viewpoints the error is the most likely to occur so that we can exclude the others. This expands the problem of quality inspection since the predefined pipeline for the inspection is broken and trajectories need to be generated based on the prediction of potential errors. We, therefore, propose to use motion-planners to generate trajectories to the desired discrete viewpoints. Different classical and heuristic approaches to motion planning already exist [19], so implementation of these planners is not the main focus of our interest. We focus on the method of experimentally choosing the most suitable one. The proposed approach combines a NN capable of detecting the error and a planner that is capable of generating trajectories on-line in order to accelerate the quality inspection process.

Motion planners are an excellent tool for finding a collision-free motion between an

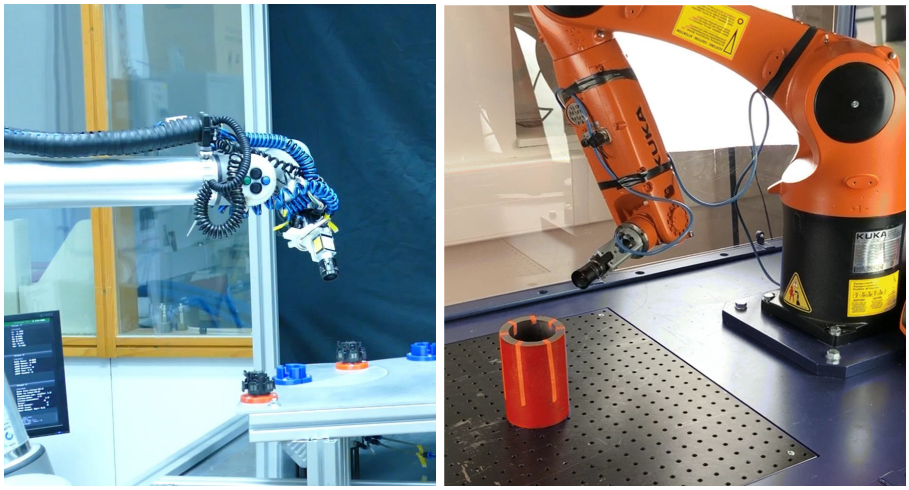


Figure 1.2: Robot inspecting the workpiece from the discrete viewpoint with the in-hand industrial camera. Right picture reprinted from [20], used under Creative Commons CC-BY license.

initial and final configuration within a specified environment [21]. However, they fail on more complicated dynamic tasks where the robot should interact with the environment - Figure 1.3. For this purpose, reinforcement learning (RL) is mainly used [22]. At its core, it relies on the robot interacting with the environment in order to adapt its behaviour completely autonomously. It was successfully applied to control complex robot systems [23], multi robot systems [24] as well as in the form of end-to-end perception-to-control policies [25]. The main drawback of this approach is that it can be a complicated and lengthy process, which requires numerous iterations, trials, and repetitions, all of which might not be safe for the robot or its immediate environment [26]. For these reasons, simulation, where data acquisition is cheaper, safer, faster and more diverse [27] is mainly used. As always, some compromise has to exist, so simulations have their drawback – inevitable modelling approximation errors [27]–[29], which result in different behaviour when applied to the real world. This difference is called the reality gap (RG) [30], and a general approach to quantify it has not been presented yet [27]. One way to overcome this problem is with transfer learning (TL), where the task/behaviour is first learned in simulation or presented by a mathematical model [31], [32] and then transferred to the real world [28], [33], [34]. This process is part of the so-called domain adaptation, where source policy provides priors to the subsequent RL of tasks [35], [36]. Since RL is based on searching for the correct parameters that describe the policy, it can suffer from sample inefficiency and the curse of dimensionality [37]. This is why DMPs or CDMPs, both capable of reducing the number of parameters for smooth trajectories, are often taken as a motion representation in RL [37], [38]. In order to reduce the search space even further, two options are available. One is to employ Principal Component Analysis (PCA) [39], which involves projecting the data onto a vector space defined by basis vectors determined by the variance of the data. Another option is to use encoder-decoder networks, which are artificial NNs that learn efficient data encoding in an unsupervised manner. The input data is passed through the layers of the NN and the layer with the smallest number of neurons, known as the latent space [40], [41], can have smaller dimensionality than the input data. While PCA provides a linear transformation of the data, encoder-decoder networks, such as autoencoder networks [42] or variational autoencoders [43], offer a nonlinear transformation, which is considered more suitable for the nonlinear nature of our world [44]. As with all

NNs, encoder-decoder networks have a strong point of being able to generalize [45]. This makes them a suitable candidate for storing the knowledge obtained in the source domain. The concept that we propose for solving the problem of autonomous learning consists of learning the complete model using imitation learning or applying RL in the search space of DMPs and storing the policy into the encoder-decoder structure NN with a latent space. When the network is fully trained and capable of performing the desired actions in the source domain, we proceed with updating only the part of the network using RL in its latent space and backpropagating with new data from the target domain [31], [46].

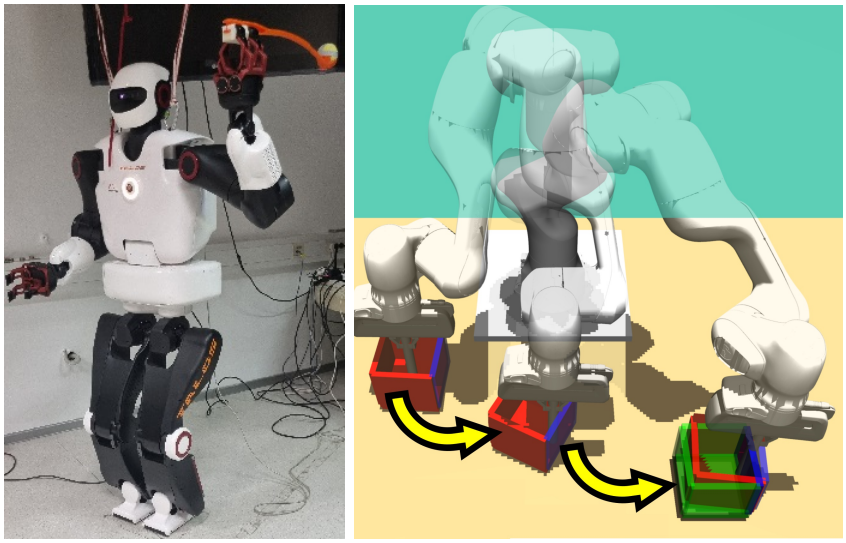
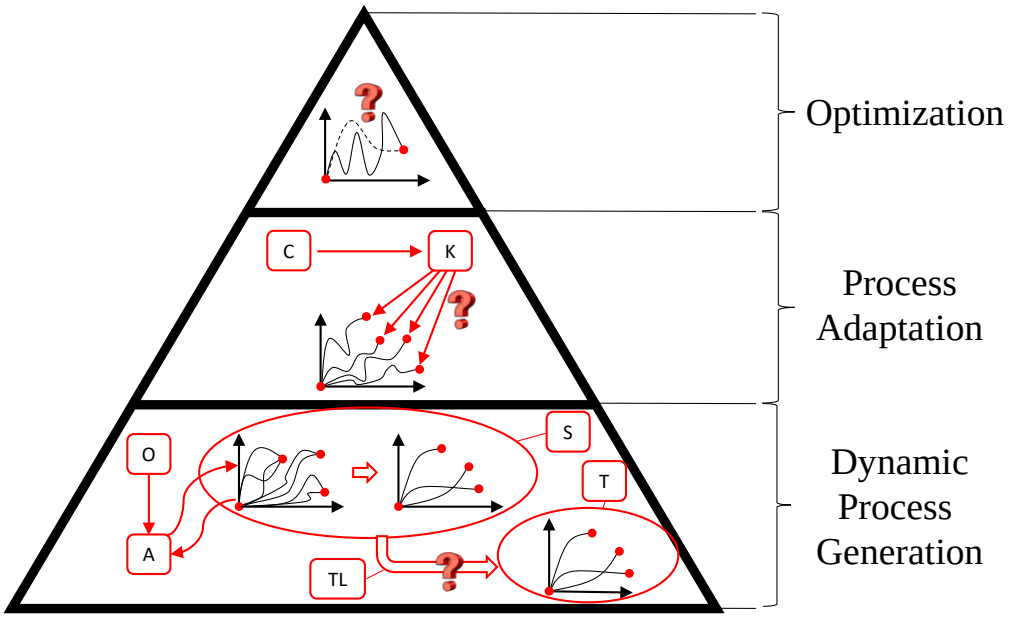


Figure 1.3: Generation of robot task behaviour with dynamic interaction between a robot and environment.

1.1 Autonomous Generation of Robot Behaviour

The overall goal of the thesis is to apply and advance methodologies of autonomous robot behaviour learning processes. In Figure 1.4, we classify robot learning with respect to its complexity. The upper part of the pyramid (optimization) shows the most straightforward case, when executable trajectory that solves a given problem is already known. This solution can be autonomously adapted/optimized in order to gain increased efficiency of the given process. The middle part of the pyramid (process adaptation) shows the case where the single solution trajectory is not known, but the overall robot task is known. Based on the parameters of the process, the task itself can be optimized. Thus, in this case, there is a big, but finite number of possible options for adaptation. As shown in Figure 1.4, they are determined, for example, based on the output of a NN as the condition (C). By having prior knowledge about the task (K), there is a certain number of possible trajectories that need to be generated. Since in the scope of this field of robot learning we are optimizing only the process and not a single trajectory, approximated trajectories are not guaranteed to be optimal. However, they can be further separately optimized using the approach from the optimization part of the pyramid. The bottom part of the pyramid (dynamic process generation) shows the most complex case of robot learning. Here, the desired outcome of some robot action is given (O). Then, through reinforcement learning that interacts with the source environment (usually simulation), a set of parameters (A) that describe the trajectory capable of satisfying the given outcome are learned. In this case, the robot

interacts with the source (simulated) environment, thus making it exceedingly challenging to correctly model or take into account all the parameters. After learning the complete policy on the simplified and modelled environment, the final step is adapting the behaviour to the target environment (T) using TL.



Optimization
- Trajectory is known but not optimal
- System behaviour does not depend on environment
- Repetitive tasks

Process Adaptation
- Process is known
- Trajectories are not known
- System behaviour does not depend on environment
- Repetitive tasks with certain level of adaptation possible based condition (C) and prior knowledge about the task (K)
- Finite number of possible outcomes

Dynamic Process Generation
- Process is known
- Trajectories are not known
- System behaviour heavily depends on environment
- Non repetitive tasks with infinite number of possibly desired outcomes (O)
- Only outcome is given
- Solution is found by searching parameters (A) in source environment (S) and transferred (TL) to target environment (T)

Figure 1.4: Pyramid graph showing the different levels of the autonomous robot behaviour generation. Top part of the pyramid shows the simplest case, when the known trajectory is optimized, middle part shows the case when the process is known but can be adapted, and the bottom part shows the case of behaviour generation from scratch and adapting it to the real-world dynamic environment.

Besides showing the classification based on complexity, the presented pyramid structure in Figure 1.4 also shows the inheriting properties of robot learning approaches. Each lower part can be additionally optimized using the top part of the pyramid. Similarly, if the robot behaviour is learnt successfully from scratch using the method from the bottom layer, it can be used in the middle layer for trajectory generation. Afterwards, these trajectories could be further optimized again using the method from the top of the pyramid.

Hereinafter, we provide an implementation example with concrete examples for each level of the pyramid given in Figure 1.4. Implementations and details about used algorithms

are given in the top-down approach, beginning with the optimization example, followed by adaptation, and concluding with dynamic task generation.

1.1.1 Trajectory optimization for higher productivity

Installing or modifying robot-supported production and quality inspection is commonly tedious and necessitates the continuous engagement of a human expert. This involvement often leads to subjective parameter specification, such as robot velocities, resulting in suboptimal outcomes. For this purpose, we introduce a new approach that allows a human expert just to select the desired points in a CAD system along the inspection path. From the selected path data, the system computes temporal parametrization of the path, which ensures the smoothness of the resulting robot trajectory for visual inspection. Our approach then combines iterative learning control (ILC) and reinforcement learning to optimize robot velocity along the path. The numerical estimate of image sharpness is used to produce the fastest possible motion without degradation of image quality. Hereinafter, we showcase a concrete implementation of trajectory optimization and provide an account of our approach. Through this example, we aim to offer insights into the practical application of trajectory optimization and the algorithms used to achieve it.

Our method assumes that a CAD model of the workpiece to be inspected is available, thus, we developed an approach that allows factory engineers to specify the desired inspection path in a CAD system in order to extract the path. Initial trajectory specification from the CAD model is a three-step process that includes path specification, time parametrization and trajectory generation. Our approach relies on the ability to easily modulate and learn the robot's velocity in a reduced number of parameters. This is one of the main features of Dynamical Movement Primitives (DMPs) [47]. A variation of DMPs, called Cartesian Space Dynamic Movement Primitives (CDMPs) [48], that except position also allow specification of Cartesian orientation, was used in our system. Initial trajectory is thus encoded in the form of CDMP with an explicitly defined velocity profile [49]. Once we have the initial trajectory, we start the optimization process. This is a two-step process where the velocity profile of the motion is learned by changing the weights that encode the velocity of CDMP. In the first step of optimization, we execute the trajectory extremely slowly and get the reference values for the image quality. After that, we execute the trajectory fast and take it as a starting point for our learning algorithm. In the proposed learning algorithm for speed modulation, we used both Iterative Learning Control (ILC) [50], [51] and a simplified version of PoWER (Policy learning by Weighting Exploration with the Returns) reinforcement learning (RL) algorithm [37]. These algorithms belong to the group of black-box algorithms that do not require the availability of analytical models that link utility functions (in our case image quality) and the robot behaviour (in our case velocity of the end-effector holding the camera). Such algorithms have been applied extensively for robot learning and policy search [22], [52], [53]. Since the analytical model is not accessible, i.e., the function relating motion to image quality is a black-box, every evaluation of this function requires the execution of visual inspection motion with a real robot and estimating the quality of images acquired during the real robot motion [54]. At the end of the learning, the resulting trajectory is slowed down only on segments where it is necessary to keep the image quality as close as possible to the reference one.

1.1.2 Adaptive process behaviour

Hereinafter, we present an example related to the middle layer of the pyramid, which pertains to the quality inspection process. This is still a scenario where the robot does not physically interact with the environment, and therefore the outcome of its actions depends

solely on the generated movement.

The process of quality control of products and semi-products in the big manufacturing lines is often done by an in-hand camera that is moved with a robot arm. This allows for checking the object from several discrete viewpoints. With complicated products, the number of necessary viewpoints can be high, creating a bottleneck in the production line. The main idea of our approach is that some of the viewpoints can be excluded from this check. We consider the relationship between the possible defects and production process parameters. This relation, although usually not intuitive, can be found using NNs. In the first part of the proposed approach, we present a NN architecture capable of predicting defects that are most likely to happen. This network is trained on data from the functional production line where inputs were readings from the sensors and outputs were manually labelled defects. Based on the error prediction of a dedicated NN (C), and having the knowledge (K) about the viewpoints that correspond to the predicted error, it is possible to check the object only from some of the pre-defined, discrete viewpoints. However, this expands the robot motion generation problem, because the robot does not follow predefined trajectories anymore, but moves between (predefined) viewpoints in the order predicted by the potential defects. Because of this, the pipeline for quality inspection is no longer always the same, and there is a huge number of possible transitions between the given viewpoints. Since all the required trajectories cannot be calculated in advance in practice, we need to use motion planners that are able to generate fast and collision-free trajectories online. As the second part of this topic, we present the existing motion planners and introduce a benchmarking process for choosing the right planner for the given process.

1.1.3 Dynamic task and process behaviour generation

Hereinafter, we present an example of the bottom level of the pyramid, which is the most complex. At this level, the robot interacts with objects in the environment, and the outcome of its actions depends not only on the generated trajectory but also on the surrounding environment. Through this example, we aim to illustrate the practical application of the concepts presented in the bottom layer of the pyramid and highlight the challenges that arise when dealing with environments that are subject to change.

Dynamic tasks are within a group of the most complicated and hard-to-engineer tasks in robotics. The main feature of this task is that a robot interacts with an environment, and it is not possible to accurately model or predict the behaviour. For this purpose, we present a methodology for obtaining the desired skill in simulation (source environment) and then adapting it to the real-world (target) environment. Since training from scratch using the existing RL techniques has been already widely addressed in research [22], [23], we only briefly introduce the possibilities of obtaining the knowledge in simulation but put the main focus on the process of transferring the knowledge to the new environment. Comparison of different methods requires vast amounts of testing. In our case, we use different simulators that act as source and target environments. Our method relies on domain adaptation using the combination of NN retraining enhanced with RL algorithms. In this approach, the initial NN is trained on the knowledge from the source environment and only then additionally adapted to the new – target environment. RL itself introduces problems when the search space is too big. Therefore, our NNs have layers with reduced size, the lowest being the latent space, where learning is done. In order to use NNs, we also need a fixed number of inputs/outputs. Because of this reason, we have an intermediate layer of encoding all our trajectories in a parametric form provided by the aforementioned DMPs. The proposed combination of reduced search space, NN retraining, and RL increases the overall performance of domain adaptation, therefore enabling application of our method to real system. In addition to this, we briefly present and benchmark two methodologies

of storing the source domain knowledge that are based on probabilistic representations of trajectories such as Probabilistic Movement Primitives (ProMPs) [55] and Probabilistic Movement Dynamic Primitives (ProDMPs) [56]. Although the principle is the same - adaptation within these networks is done in the same manner with retraining only the small part, it has been reported that these methods have achieved higher robustness to the changes from source to target environment [57].

1.2 Review of the Relevant Scientific Field

The proposed dissertation advances the current state of the art in the field of autonomous robot learning of trajectories. It presents new and advanced methods in trajectory optimization, trajectory generation and policy generation.

Optimization of robot trajectories has been the topic of many research papers. The majority of the optimization techniques rely on having known models [7]–[9]. However, in many cases, the model is not available or too complicated to formulate mathematically. One such case is the image quality when performing continuous quality inspection, where the lighting plays an important role. It is possible to approximate this model using the NN, as shown in [11], but there is no guarantee that we have considered all the possible parameters of the environment. Another approach is to use image post-processing, but this method does not increase inherent information content, and a good original image can provide more information [54]. In our method, we do not build a model but use known learning methods and reinforcement learning in order to optimize the trajectory velocity based on the image sharpness. To the best of our knowledge, no previous research addresses the issue of speed optimization while preserving image quality.

Once we have a trajectory, we are able to optimize it. However, if trajectory is unknown, we need to generate it. That is why a lot of work in current research is put into autonomous generation of trajectories [58], [59]. The inspection of finished products for quality control purposes has been identified as a common bottleneck in many industrial processes. Checking the product using the in-hand camera and a robot arm can be particularly time-consuming. In our work, we develop a full end-to-end system that is able to identify potential defects in the products based on the process parameters using a NN and accordingly generate the robotic trajectories. Although path planners are not part of our work, we present the methodology of choosing the best option from the plethora of widely available ones [17]. To our best knowledge, a fully automated process of predicting the defects and generating the trajectories has not been addressed in this manner in the previous research.

In continuation of first optimizing a trajectory, and then generating a robot trajectory, we expand our work by generating a robot trajectory that manipulates a dynamical object into desired behaviour. For this purpose, RL and particularly deep RL has achieved impressive success, but mainly in simulation environments [27], where it is safer and faster to obtain data. Due to the modelling errors [27]–[29], policies learned in simulation do not perform well when applied to the real system. This problem can be solved using TL, that has been a topic of many review papers [27], [34]. Therefore, we propose learning a complete policy in the source environment where it is relatively easy and fast to acquire data. Single instances of task are first learned separately using RL. Once we obtained successful examples of behaviour on multiple single instances, we encode this knowledge in NN that is capable of generalizing the knowledge to previously unseen instances. Further, we combine traditional domain adaptation with using the RL algorithm and NN backpropagation in order to adapt the knowledge to the target environment. The adaptation is made only on one part of the network, thus making it feasible to perform the more efficient few-shot TL.

1.3 Hypothesis

We hypothesize that:

- Autonomous learning approaches such as reinforcement learning and iterative learning control can be used to optimize the motion. This motion is faster and easier to acquire than human-calculated motion.

  Motions for quality inspection in industrial environments are manually tuned or calculated by the operator and, therefore, often suboptimal either in terms of the execution time or the image quality. Our system is capable of extracting the trajectory from the CAD model of the object and autonomously optimizing the velocity profile of the trajectory better and faster than a human operator.

- Motion planners are capable of generating the robot trajectories online.

  There is a wide range of trajectory planners available. We present the method of choosing the most suitable motion planner that is capable of generating robot trajectories for the given use-case online.

- AI can be used to accelerate the quality inspection process by predicting the possible errors.

  Potential errors in the final product are dependent on the production line parameters. Because of a high number of parameters and non-intuitive numerical values, it can be challenging for a person to notice a correlation between the error in the product and the set of parameters. NNs are capable of solving this task for us. This way, they accelerate the production line process by allowing us to exclude the product errors that are highly unlikely to happen.

- Learned robot behaviour can be effectively reused in changed environments by applying RL in order to perform TL.

  RL is capable of learning robot actions from scratch. In general, this process requires a lot of time and fails if the environment parameters are changed. However, it can be efficiently reused as the initial policy for TL.

- RL can be used to accelerate TL.

  A combination of backpropagating the new data through the NN can be combined with RL algorithms in order to achieve faster adaptation of the learned behaviour in the case when the environment parameters are changed.

1.4 Contribution of the Thesis

In this thesis, we have developed and evaluated systems for:

1. Robot trajectory optimization

   Execution of robot trajectories in industrial environments should be optimized with respect to time. We present a trajectory optimization method using the example of a quality inspection process in which the robot holds a camera and performs inspection using continuous stream of images, as published in [54]. To enable optimization, we introduce a system capable of autonomously extracting the trajectory from the CAD model of the object. This trajectory is then encoded using CDMPs [48] with an explicitly defined velocity profile [49] enabling us to reduce the dimensionality of the

problem. We have developed a model-free approach that depends only on a camera focus measure [60], in which we apply a combination of ILC [50] and an RL algorithm [37] to improve the cycle time of the quality inspection cell.

2. Motion planning based on AI-predicted process behaviour

   Checking for all potential errors for each produced part in large-scale industrial processes would create a bottleneck in the manufacturing pipeline. Therefore, we have developed an AI-based system capable of excluding the errors that are not likely to happen from the inspection process. The system is based on training an AE network with data provided from the real production line. Once the AE network is trained, we perform clustering using k-means and GMM methods in its latent space, based on a labelled dataset of defective parts. Inspecting only for the predicted defects implies different inspection process/trajectories for each part. To avoid manually generating all possible trajectories, we have implemented an approach using motion planners [58]. Since only some of them are capable of generating the trajectories in real-time, we use a benchmarking framework [59] to determine which of the planners can successfully generate trajectories for our specific use-case of inspecting parts in the car industry [18].

3. Fully autonomous learning

   In order to acquire new policies fully autonomously from scratch, we present two deterministic methods for generating examples in the source environments [61], [62], storing the obtained knowledge in NNs, and adapting it using a small amount of data from the target environment through TL techniques [31], [46]. Our main focus is on advancing TL methods to enable their application with real-life realistic number of iterations. We accelerate traditional TL methods that are based on back-propagation by applying an additional RL algorithm to the parameters of the NNs, and we demonstrate our approach on a throwing action using a robot arm [31], and using a humanoid robot [46]. Additionally, since there are recent publications that report probabilistic approaches are more robust to changes in the environment [57], we present and compare two NN architectures capable of predicting the distribution of trajectories, showing how to offer more diversity while maintaining a lower error [63].

Chapter 2

Trajectory Optimization for Higher Productivity

The most widely used and, at the same time lowest by its complexity, technique in autonomous robot behaviour generation is the trajectory optimization. This is the reason why in our pyramid that classifies robot generation, it is put on the top - Figure 2.1. In this case, the trajectory that is able to execute the task successfully is already known, but there is still space for further improvement in the sense of making it faster, smoother or more energy efficient. All the other techniques, that are by the complexity ranked higher in our pyramid, are able to autonomously generate non-optimal trajectory but further we need to use trajectory optimization technique in order to improve it. The main characteristics of these techniques are that the trajectory is known but not optimal, system behaviour does not depend on the environment, and it is mainly used for repetitive industrial tasks.

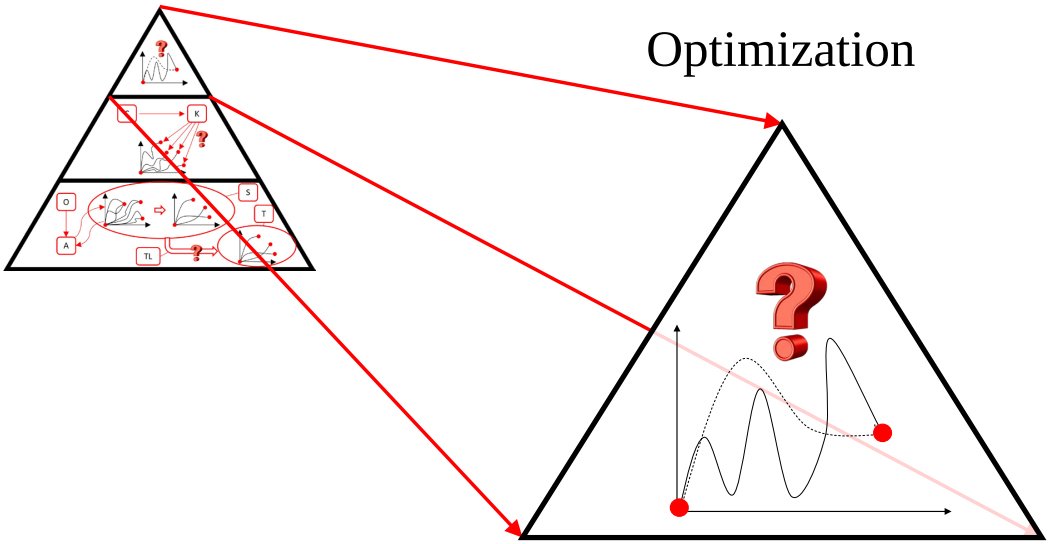


Figure 2.1: Trajectory optimization is set to be on the top of our classification pyramid since it is the simplest by complexity and the most widely used.

Optimization is mainly used in industrial environments where huge number of products need to be manufactured in the limited time. Moreover, since the product types are being changed more frequently, it is also important that modification of the production line can

be performed as quickly as possible. It is a known problem that setting up or modifying any production line sections can be tedious and requires human expert engagement. The same stands also for the sections where the quality control is performed. The performance is, therefore, dependent on the skill of an expert being in charge of setting up robot velocities and prone to having suboptimal results. To address this issue, in this chapter, we introduce a new method for specifying visual inspection trajectories based on CAD models of the workpieces to be inspected. Our goal is to enable human experts to select the desired points on the inspection path within a CAD system and making the rest of the process fully automated. The system computes the temporal parametrization of the path to ensure smoothness of the resulting robot trajectory. We also propose a new learning method that combines iterative learning control (ILC) and reinforcement learning (RL) to optimize robot speed along the specified path. The method takes into account the image quality and produces the fastest possible motion that does not compromise the quality of the image.

2.1 Methodology

Our approach involves optimizing the robot trajectory through a model-free method, which relies on the system's ability to adjust the velocity of the trajectory. The optimization is based on an image quality measure, which serves as the reward function for evaluating the performance of the executed trajectory. We first introduce the image quality measure to quantify the performance of the trajectory. Then, we describe how we define the motion trajectories in order to achieve quick and easy velocity modulation in a small number of parameters. Finally, we provide a detailed explanation of the optimization procedure used.

2.1.1 Image quality measures

Visual quality inspection requires sharp, well-focused images. Here, we analyze possible measures that can be applied to measure image quality.

The first step in our procedure is to determine the distance to the workpiece at which the in-hand camera captures well-focused images. This is necessary to ensure good and constant image quality because most industrial cameras on the market do not provide autofocus. For the cameras that do provide autofocus ability, there is little information about how the focus is determined [15]. For cameras with a fixed focal length, a sharp image is obtained only if the camera is placed at the proper distance from the observed workpiece. To automate this process and avoid a subjective sharpness assessment by the engineer, Bevec et al. [15] developed an approach to automatically place the in-hand camera at an appropriate distance from the workpiece so that the workpieces' surface is in focus. The proposed autofocus algorithm uses the distinct bell-shape characteristics of the squared gradient focus measure, which has the highest value when the picture is sharp. By moving the hand-held camera perpendicularly to the workpiece, towards and away from the workpiece, the robot finds the appropriate distance that results in the sharpest image, with results surpassing manual settings [15].

Once the proper distance for obtaining sharp images has been determined, we need to make sure that the image quality is preserved as the robot moves the camera along the workpiece to be inspected, i.e., we need to prevent motion blur. There are many quality measures (also called focus measures) proposed in the literature to estimate the sharpness of the acquired images. They are based either on computing the first or second order image derivatives, image histograms, correlation and data compression [60]. The methods that employ derivatives use different operators to compute them, such as Sobel operators

(horizontal, vertical, combined), Scharr operators, Laplacian, and others. For example, the formula to compute the squared magnitude of the horizontal gradient is given by:

$$\phi_h = \sum_{x=0}^{W-1} \sum_{y=0}^{H-2} \left(I(x, y+1) - I(x, y)\right)^2, \tag{2.1}$$

where H and W are respectively the height and width of the image and $I(x, y)$ are the intensity values at pixel (x, y). Similarly, the squared magnitude of the vertical gradient is given by:

$$\phi_v = \sum_{x=0}^{W-2} \sum_{y=0}^{H-1} \left(I(x+1, y) - I(x, y)\right)^2. \tag{2.2}$$

To test which image quality measure is the most appropriate, i. e., best for the conditions that typically arise during the quality inspection, we set up an example robotic cell for visual quality inspection. We programmed the robot equipped with an in-hand camera to move along a straight line above a flat textured surface at the fixed distance determined by the above-described autofocus algorithm. The trajectory was programmed as a quintic polynomial with the given initial and final position and the initial and final velocities and accelerations set to zero [64]. The duration of movement was set to 3, 20, or 60 seconds, which resulted in trajectories with different speeds.

Figure 2.2 shows the relative value of the computed focus measures with respect to the initial values, i. e.

$$\frac{\phi(t/T_e)}{\phi(0)}, \tag{2.3}$$

where T_e is the duration of the motion execution. The values in Figure 2.2 were obtained by normalizing the results of the focus measures with the number of pixels in the image. The durations of movements were normalized to 1. Figure 2.2 shows that the focus measure values decrease with the increased speed, as the values in plot c), which present the results gathered during fast camera motion, are much lower. Fast motion, therefore, introduces more image blur, which decreases the quality of the acquired images. Since focus measures are noisy, we used filtered values in learning. Because learning is performed off-line, we filtered them with a zero-phase low-pass filter.

Next, we computed the relative values of the focus measures for the fastest (3 seconds) and slowest (60 seconds) camera motion, i.e.

$$\phi_r = \frac{\phi_3\left(\tilde{t}\right)}{\phi_{60}\left(\tilde{t}\right)}. \tag{2.4}$$

where $0 \leq \tilde{t} \leq 1$ denotes the normalized duration of motion for each movement. The results for the top 10 focus measures (see Figure 2.2) are shown in Figure 2.3. We can see that the squared horizontal gradient focus measure (marked Gradient H) is the most reactive to the changes of velocity. Note that the values would be the same for the squared vertical gradient if the camera were rotated by 90°. We therefore used the sum of squared magnitudes of vertical (see Eq. (2.2)) and horizontal (see Eq. (2.1)) gradient focus measures to evaluate the sharpness of the image.

2.1.2 Specification of motion trajectories for visual quality inspection

We assume that a CAD model of the workpiece to be inspected is available. Here, we describe a three step process to specify the initial trajectory for visual quality inspection. It should be noted that CAM software packages can provide trajectories for typically 5-axes

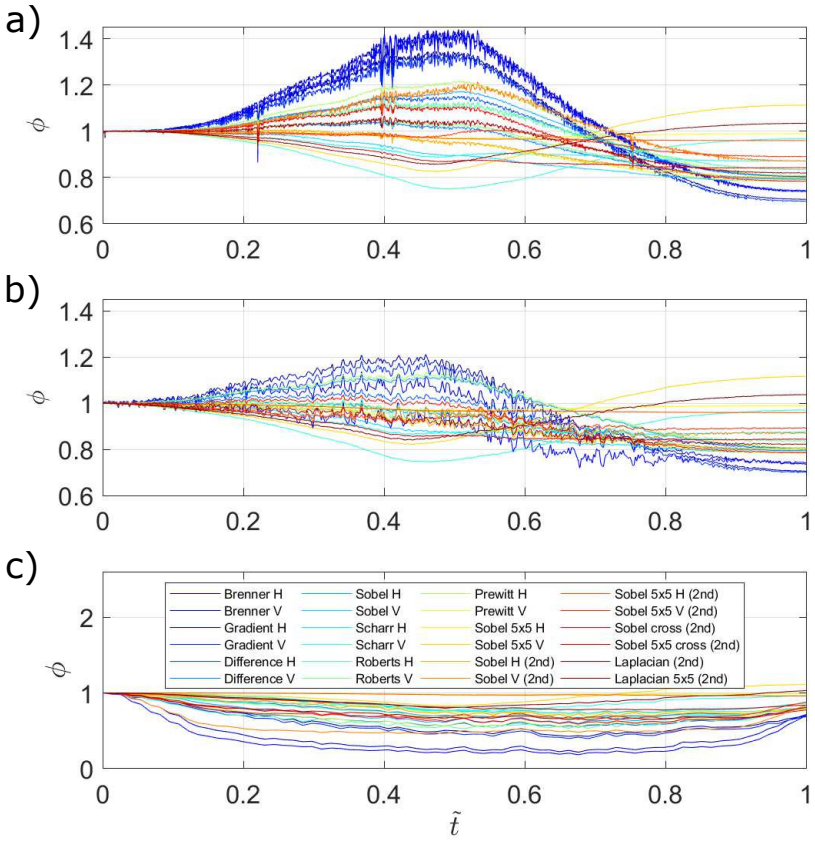


Figure 2.2: Focus measures ϕ at different speeds. Plot a): 60 seconds motion. Plot b): 20 seconds motion. Plot c): 3 seconds motion. The time axis is normalized to 1 for all graphs.

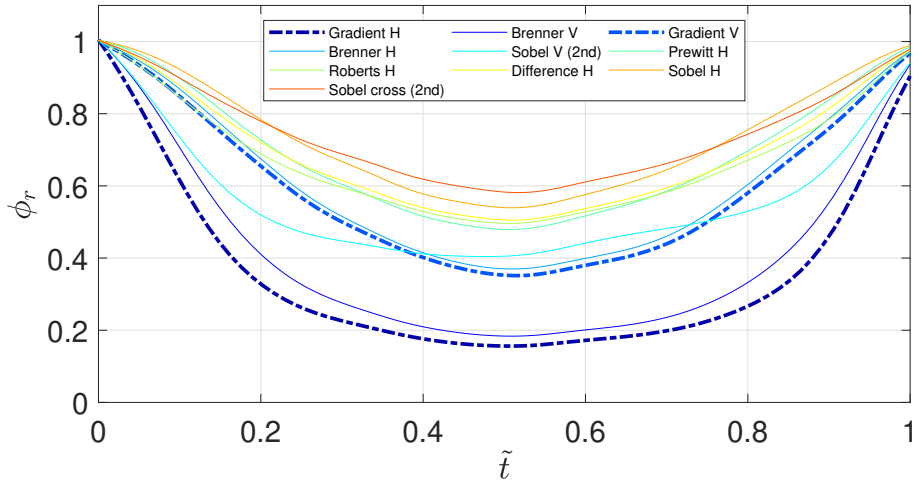


Figure 2.3: Relative change of the filtered focus measures for 10 different measures as a function of normalized time.

CNC machines. However, robot motion for visual quality control has to be fully specified in all 6 dimensions of the camera motion. Additionally, CAM-software-generated paths

usually do not take care of discontinuities in the angular velocity that arise due to the sudden changes in the curvature of the observed surface. We have therefore implemented our own temporal parametrization of paths exported directly from a CAD system. Besides generating smooth initial trajectories for visual quality inspection, this way we also avoid high costs associated with the purchase of plug-ins needed for trajectory generation from CAM systems.

Path specification

To specify a path, the CAD model of the workpiece is imported into the CAD system and a human expert interactively selects 3-D points on the workpiece using a graphical user interface of the CAD system. Standard CAD systems such as PCT Creo [65] can be used for this purpose. The human expert selects the start and end point of each linear segment on the surface of the workpiece of the CAD system or sketches the path in 2D. The CAD system then projects the specified linear or sketched 2D path onto the surface of the workpiece. Once projected onto the surface of the workpiece, the path is sampled by the CAD system, where the expert defines the sampling by specifying the desired Euclidean distance between consecutive points.

The orientations along the path are determined by gathering the normals to the workpiece surface $\boldsymbol{n} \in \mathbb{R}^3$ and tangents of the path $\boldsymbol{t} \in \mathbb{R}^3$. Both can be computed by the CAD system at each sampling point on the path. Since the path lies on the surface of the workpiece, its tangent is orthogonal to the workpiece surface normal. Thus, at each sampling point, a 3-D rotational matrix specifying the orientation of the camera can be defined as follows

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{t} & \boldsymbol{t} \times \boldsymbol{n} & -\boldsymbol{n} \end{bmatrix} \in \mathbb{R}^{3\times 3}. \tag{2.5}$$

Note that the camera orientation around the optical axis, which is parallel to the surface normal of the workpiece but in the opposite direction, is often not important in visual quality inspection tasks and could be selected arbitrarily. The above approach using Eq. (2.5) is reasonable as it directs the camera perpendicularly to the workpiece (provided the z axis of the tool center point is aligned with the camera optical axis). In addition, one of the image axes is aligned with the direction of robot motion (provided the camera coordinate system is aligned with the robot base coordinate system). The rotation matrices representing orientation are transformed to quaternions before export.

The result is a dense sequence of positions and orientations on the desired path together with surface normals (all exported from the CAD system)

$$\{\{\tilde{\boldsymbol{p}}_{j,k}, \tilde{\boldsymbol{q}}_{j,k}, \boldsymbol{n}_{j,k}\}_{k=0}^{K_j}\}_{j=1}^{K}. \tag{2.6}$$

They are given in the workpiece coordinate system. Here $\tilde{\boldsymbol{p}}_{j,k} \in \mathbb{R}^3$ are the positions, $\tilde{\boldsymbol{q}}_{j,k} \in \mathbb{R}^4$ the orientations specified as unit quaternions, and $\boldsymbol{n}_{j,k} \in \mathbb{R}^3$ the unit surface normals at points $\tilde{\boldsymbol{p}}_{j,k}$. The unit quaternions $\tilde{\boldsymbol{q}}_{j,k}$ are computed by transforming the rotation matrices $\boldsymbol{R}_{j,k}$ generated using Eq. (2.5) at $\tilde{\boldsymbol{p}}_{j,k}$. Every subsequence $\{\tilde{\boldsymbol{p}}_{j,k}, \tilde{\boldsymbol{q}}_{j,k}, \boldsymbol{n}_{j,k}\}_{k=0}^{K_j}$ consists of twice continuously differentiable positions and orientations whereas transitions between subsequences (segments) can be discontinuous. These data are determined in the CAD system and are exported in STEP format.

Next, we lift the inspection paths to the observation distance $d_f \in \mathbb{R}$ determined by the autofocus procedure described in [15] and transform the lifted paths to the robot coordinate system. The following transformation is applied to every subsequence, $j = 1, \ldots, K$,

$$\{\boldsymbol{p}_{j,k}, \boldsymbol{q}_{j,k}\}_{k=0}^{K_j} = \{\boldsymbol{q}_w * (\tilde{\boldsymbol{p}}_{j,k} + d_f\, \boldsymbol{n}_{j,k}) * \bar{\boldsymbol{q}}_w + \boldsymbol{t}_w,\, \boldsymbol{q}_w * \tilde{\boldsymbol{q}}_{j,k}\}_{k=0}^{K_j}, \tag{2.7}$$

where $\boldsymbol{t}_w \in \mathbb{R}^3$ and $\boldsymbol{q}_w \in \mathbb{R}^4$ are the position and orientation of the workpiece in the robot base coordinate system. The pose of the workpiece in the robot base coordinate system is usually determined by vision, where the camera is calibrated to the robot base coordinate system. This way we obtain the desired paths $\{\boldsymbol{p}_{j,k}, \boldsymbol{q}_{j,k}\}_{k=0}^{K_j}$ expressed in the robot base coordinate system.

Evolution of time along the inspection path

In the second step, we specify a suitable evolution of time for the desired inspection trajectory. Given a smooth sequence of poses $\{\boldsymbol{p}_{j,k}, \boldsymbol{q}_{j,k}\}_{k=0}^{K_j}$ on the robot path, we can estimate the arc length along the path as follows

$$s_{j,k} = s_{j,k-1} + \|\boldsymbol{p}_{j,k} - \boldsymbol{p}_{j,k-1}\|, \ k = 1, \ldots, K_j, \ s_{j,0} = 0. \tag{2.8}$$

For orientation, we estimate the angle length instead of the arc length

$$\xi_{j,k} = \xi_{j,k-1} + 2\|\log(\boldsymbol{q}_{j,k} * \overline{\boldsymbol{q}}_{j,k-1})\|, \ k = 1, \ldots, K_j, \ \xi_{j,0} = 0. \tag{2.9}$$

Thus the overall arc length of the path is equal to $L_j = s_{j,K_j}$ while the overall angle length of the path is equal to $\zeta_j = \xi_{j,K,j}$. We assume that $(\boldsymbol{p}_{j,k}, \boldsymbol{q}_{j,k}) \neq (\boldsymbol{p}_{j,k-1}, \boldsymbol{q}_{j,k-1})$, $k = 1, \ldots, K_j$. If any two consecutive locations are the same, we delete the duplicates as they do not contribute to the progress along the path.

To specify the evolution of time, we parametrize the weighted sum of arc and angle length with time. First the desired duration of inspection movement is calculated as follows

$$T_j = \alpha(L_j + \beta\zeta_j), \tag{2.10}$$

where $\alpha, \beta > 0$ are suitable constants, which need to be carefully selected by the inspection path designer. Alternatively, the designer can specify the times T_j to traverse each path segment instead of α. We then time-parametrize the weighted sum of arc and angle length, i. e.

$$r(t) = s(t) + \beta\xi(t), \tag{2.11}$$

with a quintic polynomial [64], setting $r(0) = \dot{r}(0) = \ddot{r}(0) = \dot{r}(T_j) = \ddot{r}(T_j) = 0$, $r(T_j) = L_j + \beta\zeta_j$. Note that the resulting polynomial r is monotonically strictly increasing on the interval $(0, T_j)$, i.e. $\dot{r}(t) > 0, \ 0 < t < T_j$. For any given arc length $s_{j,k}$ calculated in Eq. (2.8) and angle length $\xi_{j,k}$ calculated in Eq. (2.9), we can therefore easily calculate the times at which they are reached by inverting function $r(t)$

$$t_{j,k} = r^{-1}(s_{j,k} + \beta\xi_{j,k}), \ k = 0, \ldots, K_j. \tag{2.12}$$

Bisection or Newton's method can be used for this purpose. The time sequence $\{t_{j,k}\}_{k=0}^{K_j}$ is guaranteed to be monotonically strictly increasing because $(\boldsymbol{p}_{j,k}, \boldsymbol{q}_{j,k}) \neq (\boldsymbol{p}_{j,k-1}, \boldsymbol{q}_{j,k-1})$, $k = 1, \ldots, K_j$.

Trajectory generation and encoding

In the third step, we encode the trajectory into a sequence of speed-scaled Cartesian Dynamic Movement Primitives (CDMPs) [48]. For this purpose, we must first calculate the linear and angular velocities and accelerations.

The time derivatives of $\boldsymbol{p}_{j,k}$, i.e. linear velocities, and angular velocities $\boldsymbol{\omega}_{j,k}$, can be computed using numerical differentiation

$$\begin{aligned} \dot{\boldsymbol{p}}_{j,k} &= \frac{\boldsymbol{p}_{j,k} - \boldsymbol{p}_{j,k-1}}{t_{j,k} - t_{j,k-1}} \\ \boldsymbol{\omega}_{j,k} &= \frac{2\log(\boldsymbol{q}_{j,k} * \overline{\boldsymbol{q}}_{j,k-1})}{t_{j,k} - t_{j,k-1}} \end{aligned}, \ k = 1, \ldots, K_j - 1. \tag{2.13}$$

By design we have $\dot{\boldsymbol{p}}_{j,0} = \dot{\boldsymbol{p}}_{j,K_j} = \boldsymbol{\omega}_{j,0} = \boldsymbol{\omega}_{j,K_j} = 0$ as the robot should be at rest at the beginning and the end of each inspection segment. Finally, the linear and angular accelerations are computed by numerical differentiation

$$\begin{aligned} \ddot{\boldsymbol{p}}_{j,k} &= \frac{\dot{\boldsymbol{p}}_{j,k} - \dot{\boldsymbol{p}}_{j,k-1}}{t_{j,k} - t_{j,k-1}} \\ \dot{\boldsymbol{\omega}}_{j,k} &= \frac{\boldsymbol{\omega}_{j,k} - \boldsymbol{\omega}_{j,k-1}}{t_{j,k} - t_{j,k-1}} \end{aligned}, \; k = 1, \ldots, K_j - 1. \tag{2.14}$$

By design we have $\ddot{\boldsymbol{p}}_{j,0} = \ddot{\boldsymbol{p}}_{j,K_j} = \dot{\boldsymbol{\omega}}_{j,0} = \dot{\boldsymbol{\omega}}_{j,K_j} = 0$. The data calculated in Eqs. (2.7), (2.13) and (2.14) can then be used to compute a sequence of speed-scaled CDMPs.

A Speed-Scaled Cartesian space Dynamic Movement Primitive (CDMP) consists of two parts: the position and orientation trajectory. The position trajectory is specified in the same way as in standard DMPs [47]. The orientation trajectory, however, is specified by unit quaternions. Since unit quaternions do not form a linear vector space, the orientational part of the CDMP requires special treatment when computing its parameters and for its integration.

The CDMP is based on two dynamic systems: phase system and transformation system. The phase system is used to avoid explicit dependency of the trajectory on time and for synchronizing the positional and orientational degrees of freedom along the trajectory. The phase dynamics is defined as:

$$\nu(x)\dot{x} = -\alpha_x x, \tag{2.15}$$

where $\nu(x) > 0$ is a speed scaling function defined in Eq. (2.23), α_x is a positive constant and x is the phase that starts at 1 and converges to 0 as the trajectory goal is reached.

The transformation system for position trajectory $\boldsymbol{p}$ and orientation trajectory $\boldsymbol{q}$ is specified with the following differential equations:

$$\nu(x)\dot{\boldsymbol{z}} = \alpha_z(\beta_z(\boldsymbol{g}^p - \boldsymbol{p}) - \boldsymbol{z}) + \boldsymbol{f}_p(x), \tag{2.16}$$

$$\nu(x)\dot{\boldsymbol{p}} = \boldsymbol{z}, \tag{2.17}$$

$$\nu(x)\dot{\boldsymbol{\eta}} = \alpha_z\left(\beta_z 2\log\left(\boldsymbol{g}^o * \overline{\boldsymbol{q}}\right) - \boldsymbol{\eta}\right) + \boldsymbol{f}_o(x), \tag{2.18}$$

$$\nu(x)\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{\eta} * \boldsymbol{q}, \tag{2.19}$$

where $\boldsymbol{g}^p \in \mathbb{R}^3$ is the goal position and $\boldsymbol{g}^o \in \mathbb{R}^4$ the goal orientation of the movement. In a CDMP, the orientation is represented by unit quaternions $\boldsymbol{q} \in \mathbb{R}^4$. The parameters $\boldsymbol{z},\ \boldsymbol{\eta} \in \mathbb{R}^3$ denote the scaled linear and angular velocity ($\boldsymbol{z} = \nu(x)\dot{\boldsymbol{p}}$, $\boldsymbol{\eta} = \nu(x)\boldsymbol{\omega}$). For more details on quaternion product $*$, conjugation $\overline{\boldsymbol{q}}$, and the quaternion logarithm $\log(\boldsymbol{q})$, see [48]. The nonlinear functions $\boldsymbol{f}_p,\ \boldsymbol{f}_o : \mathbb{R} \mapsto \mathbb{R}^3$, also called forcing terms, are defined as

$$\boldsymbol{f}_p(x) = \boldsymbol{D}_p \frac{\sum_{i=1}^N \boldsymbol{w}_i^p \Psi_i(x)}{\sum_{i=1}^N \Psi_i(x)} x, \tag{2.20}$$

$$\boldsymbol{f}_o(x) = \boldsymbol{D}_o \frac{\sum_{i=1}^N \boldsymbol{w}_i^o \Psi_i(x)}{\sum_{i=1}^N \Psi_i(x)} x. \tag{2.21}$$

where the weights $\boldsymbol{w}_i^p,\ \boldsymbol{w}_i^o \in \mathbb{R}^3,\ i = 1, \ldots, N$, respectively encode the position and orientation parts of the trajectory and N is equal to the number of radial basis functions that are used to encode the trajectory. They have to be learned, for example directly from an input Cartesian trajectory $\{\boldsymbol{p}_k, \boldsymbol{q}_k, \dot{\boldsymbol{p}}_k, \boldsymbol{\omega}_k, \ddot{\boldsymbol{p}}_k, \dot{\boldsymbol{\omega}}_k, t_k\}_{k=0}^K$. The scaling matrices $\boldsymbol{D}_p,\ \boldsymbol{D}_o \in \mathbb{R}^{3\times 3}$ can be set to $\boldsymbol{D}_p = \boldsymbol{D}_o = \boldsymbol{I}$. Other possibilities are described in [48]. The nonlinear forcing terms are defined as a linear combination of radial basis functions Ψ_i

$$\Psi_i(x) = \exp\left(-h_i\left(x - c_i\right)^2\right), \tag{2.22}$$

where c_i are the centers and h_i the widths of the radial basis functions. The distribution of weights can be set to $c_i = \exp\left(-\alpha_x \frac{i-1}{N-1}\right)$, $h_i = \frac{1}{(c_{i+1} - c_i)^2}$, $i = 1, \ldots, N$, $h_N = h_{N-1}$ [66]. The goal position and orientation are usually set to the final position and orientation on the desired trajectory, i. e. $\boldsymbol{g}^p = \boldsymbol{p}_{t_K}$ and $\boldsymbol{g}^o = \boldsymbol{q}_{t_K}$. A more detailed CDMP description and auxiliary math are provided in [48].

The temporal scaling function $\nu(x)$ defines the speed profile of the movement. It is defined as a weighted combination of kernel functions

$$\nu(x) = \tau + \frac{\sum_{i=1}^{M} w_i^\nu \Psi_i(x)}{\sum_{i=1}^{M} \Psi_i(x)}. \tag{2.23}$$

Here $\tau > 0$ is the time constant usually set to the initial desired duration of motion. M specifies the number of kernel functions Ψ_i, which are defined as in Eq. (2.22). The parameters $w_i^\nu \in \mathbb{R}$ are initially set to zero. They are included to enable nonuniform speed scaling along the trajectory. Note that phase x is used also to drive the scaling of speed.

In the case of discontinuities along the consecutive trajectories, such as for example around a perpendicular edge, the robot needs to move from the end of one visual quality inspection trajectory to the beginning of the next one. Visual quality inspection is not applied during such transition movements. Hence the robot can move at a maximal velocity between the end and start configuration along the shortest path between them. One can apply trapezoidal velocity profiles with maximal joint velocities [64] to generate the transition movements.

The next step is to optimize the duration of robot motion for visual quality inspection. We can either optimize each trajectory segment separately or we can combine some of the segments into one CDMP if there are no spatial discontinuities between them. The speed of the resulting CDMPs should then be optimized to enable quick but at the same time highly precise visual quality inspection. This optimization process is described in the following subsection.

2.1.3 Optimization objective

Once the initial visual quality inspection motion has been specified as a sequence of smooth trajectories encoded by a sequence of speed-scaled CDMPs, we can proceed with the optimization of the speed of robot motion. The aim is to minimize the time of motion along each visual inspection path segment without degrading the quality of images acquired while the robot moves the hand-held camera compared to images acquired while the camera is static. This is equivalent to determining the highest speed of motion that does not degrade the image quality.

We proceed with the formulation of the optimization problem to compute such a motion. As stated above, the execution time decreases if the speed of motion increases. We can alter the speed of a CDMP without changing its spatial course by modifying the temporal scaling factor ν defined in Eq. (2.23), i.e. by modifying the weights w_i^ν, $i = 1, \ldots, M$. Thus the optimization problem can be formulated as follows:

$$\begin{aligned} &\min_{\{w_i^\nu\}_{i=1}^M} \; T_e \\ \text{s.t.} \quad &|y_d(x) - y(x)| \leq \varepsilon, \; \forall x, \\ &0 < \nu(x), \; \forall x, \end{aligned} \tag{2.24}$$

where $T_e > 0$ is the execution time, i.e. the time needed to move the camera along the complete visual inspection path, x denotes the phase of motion computed by integrating

the CDMP Eq. (2.15), and $\varepsilon > 0$ is a small constant that compensates for the system noise and specifies the allowed deviations from the ideal image quality. Image quality measure y is estimated using the following formula

$$y(x) = \phi_h(x) + \phi_v(x), \tag{2.25}$$

where $\phi_h(x)$ and $\phi_v(x)$ are defined as in Eq. (2.1) and (2.2), respectively. Other suitable measures for image quality could also be employed. y_d in optimization problem (2.24) denotes the ideal image quality measure obtained when the camera does not move. Since the mathematical model specifying the relationship between the speed profile ν on the one side and execution time T_e and image quality measure y on the other side is not known, we can only apply black-box optimization algorithms [53] to solve this optimization problem.

Due to the lack of models, every iteration of the optimization process requires us to execute the real robot motion along the visual inspection path. Among the possible black-box optimization algorithms that could be utilized to solve the optimization problem (2.24), we propose to apply a two-step process, starting with Iterative Learning Control (ILC) and followed by a RL method based on policy search. The former exploits feedback error control law and converges faster in general, while the latter can deal with more general cost functions and to some extent avoid local minimum due to the randomized search process. Initial results on ILC application were presented in [67].

2.1.4 Optimizing the speed of motion by iterative learning control (ILC)

ILC is a technique for improving the transient response and tracking the performance of processes, machines, equipment, or systems that execute the same trajectory, motion, or operation over and over [50]. The main idea of ILC is to iteratively modify the input into the system until the output has been improved.

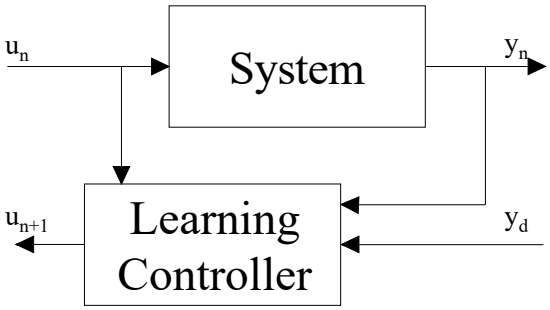


Figure 2.4: Basic ILC approach scheme.

Figure 2.4 illustrates its basic operational principle. Input and output signals are defined in the time interval $t \in [0, T]$. Subscript n indicates the trial or repetition number. When input signal $u_n(t)$ is applied to the system, it produces the output $y_n(t)$. Based on the difference between the actual and the desired output, $e_n(t) = y_d(t) - y_n(t)$, and the corresponding input $u_n(t)$, the new modified input signal is calculated as

$$u_{n+1}(t) = Q[u_n(t) + \lambda e_n(t)], \tag{2.26}$$

where $\lambda > 0$ is the gain that defines the size of the update step and Q is the low-pass filter applied to smoothen the signal.

The scheme described above is often referred to as causal ILC. Further variations of the ILC approach are discussed in [51]. Hereinafter, we continue with the application of Iterative Learning Control (ILC) to speed up robot motion along the path for visual quality inspection.

With CDMPs, nonlinear speed scaling can be achieved by modifying the weights of the speed scaling factor ν defined in Eq. (2.23). This is possible because ν modifies the speed of motion specified by integrating Eqs. (2.15) – (2.19) without changing the spatial course of motion. ILC is applied along the phase of motion instead of time. This is important because ILC can only be applied if samples are properly aligned between iteration steps. In terms of expressions explaining the casual ILC, the control scheme shown in Figure 2.4, and by replacing time with phase, the ILC control signal is specified as $u(x) = \nu(x)$, while the ILC output signal is computed as image quality y specified in Eq. (2.25).

The learning process is initiated by gathering the desired values $y_d(x)$ for image quality, as defined in Eq. (2.25). This is done by moving the robot arm holding the camera along the desired inspection path at a slow speed that does not affect the quality of the captured images. Namely, high speed motion could result in blurred images. We can achieve this by setting τ in Eq. (2.23) to a high positive value τ_{slow} and the weights to zero. The image quality is then computed at camera frequency (typically 30 Hz). We obtain $\{y_d(t_p)\}_{p=1}^{P}$, where $y_d(t_p) = \phi_h(t_p) + \phi_v(t_p)$ and P is the number of captured images during motion execution. To align the measurements captured at different speeds, we need to sample the computed image quality values along the phase of motion. Using Eq. (2.15) and $x(0) = 1$, these values can be sampled as follows in the phase domain

$$\{y_d(x_p)\}_{p=1}^{P},\ x_p = x(t_p) = \exp\left(-\frac{\alpha_x}{\tau_{\text{slow}}} t_p\right). \tag{2.27}$$

ILC iteratively adapts the speed of robot motion. Thus at each iteration step, a different number of images is captured and a different number of image quality measures is calculated. We denote the times at which the images are captured in the n-th iteration step by $t_{n,p}$, $p = 1, \ldots, P_n$. The newly computed image quality measures can be aligned with the desired quality measures from Eq. (2.27) by computing the phases $x_{n,p}$, along the executed trajectory, where $x_{n,p}$ are computed by integrating Eq. (2.15). We now have the problem that the newly computed phases $x_{n,p}$ do not exactly match the phases x_p at which the desired quality measures were obtained. This problem is resolved by cubic interpolation of the newly acquired image quality measures $\{y(x_{n,p})\}_{p=1}^{P_n}$ and computing the interpolated values at x_p, $p = 1, \ldots, P$. We denote these newly computed values as

$$\{y_n(x_p)\}_{p=1}^{P}. \tag{2.28}$$

To initialize the ILC algorithm, we set the initial ILC input signal to $u_0 = \nu_0 = \tau_{\text{fast}}$, $\tau_{\text{slow}} > \tau_{\text{fast}} > 0$, $w_i^{\nu} = 0, \forall i$. τ_{fast} is set to a small value so that the initial robot motion is too fast to capture high quality images. Thus the speed of motion should be decreased – which is equivalent to increasing $\nu(x)$ – until the motion is slow enough to capture good quality images. Following Eq. (2.26) and using image quality measure as output signal, at each step the updated input signal (before filtering) is given as

$$\nu_n(x_p) + \lambda(y_d(x_p) - y_n(x_p)). \tag{2.29}$$

The following system of linear equations can then be solved to update the weights of the speed profile:

$$\boldsymbol{X}\boldsymbol{w}_{n+1}^{\nu} = \begin{bmatrix} \nu_n(x_1) + \lambda(y_d(x_1) - y_n(x_1)) - \tau \\ \vdots \\ \nu_n(x_P) + \lambda(y_d(x_P) - y_n(x_P)) - \tau \end{bmatrix}, \tag{2.30}$$

Algorithm 2.1: Procedure for optimizing the speed profile for visual quality inspection by ILC.

procedure LearnSpeedProfile
  set $w_{i,1}^{\nu} = 0,\ i = 1, \ldots, M$;
  record $\{y_d(x_p)\}_{p=1}^{P}$ for slow motion with high $\tau = \tau_{\text{slow}} > 0$;
  set τ for fast motion to small positive $\tau = \tau_{\text{fast}}$;
  set $n = 0$
  repeat
    increase n;
    execute motion with current $\{w_{i,n}^{\nu}\}_{i=1}^{M}$ by integrating
      Eqs. (2.15) – (2.19) and gather image quality measures $\{y_n(x_p)\}_{p=1}^{P}$;
    compute $\{w_{i,n+1}^{\nu}\}_{i=1}^{M}$ by solving Eq. (2.30)
  until $\sum_{p=1}^{P} y_n(x_p) > \gamma \sum_{p=1}^{P} y_d(x_p)$

where

$$\boldsymbol{w}_{n+1}^{\nu} = \begin{bmatrix} w_{1,n+1}^{\nu} \\ \vdots \\ w_{M,n+1}^{\nu} \end{bmatrix},\ \boldsymbol{X} = \begin{bmatrix} \dfrac{\psi_1(x_1)}{\sum_{i=1}^{M} \psi_i(x_1)} & \cdots & \dfrac{\psi_M(x_1)}{\sum_{i=1}^{M} \psi_i(x_1)} \\ \vdots & \ddots & \vdots \\ \dfrac{\psi_1(x_P)}{\sum_{i=1}^{M} \psi_i(x_P)} & \cdots & \dfrac{\psi_M(x_P)}{\sum_{i=1}^{M} \psi_i(x_P)} \end{bmatrix}.$$

The ILC update specified by Eq. (2.26) contains a low-pass filter denoted by Q. This filter is not included in Eq. (2.29) but is implemented in the proposed approach by computing the update as a solution to the system of linear equations (2.30). Since there are always more measurements (P) than basis functions (M), this system is overdetermined, which results in smoothing of the signal $u = \nu$, thus effectively implementing the low-pass filter Q.

The goal of learning is to generate a new speed profile that is as fast as possible without significantly diminishing the desired image quality measure y_d. Thus the learning process is halted once the overall image quality measure becomes similar to the desired quality measure, i. e.

$$\sum_{p=1}^{P} y_n(x_p) \approx \gamma \sum_{p=1}^{P} y_d(x_p),\ 0 < \gamma < 1. \tag{2.31}$$

Parameter γ needs to be specified by a user. The algorithm summarizing the above-described procedure is provided in Algorithm 2.1.

Note that in the proposed approach, we start with a trajectory that is too fast to capture high quality images, thus the initial image quality values are low. ILC then increases $\nu(x)$ to slow down the robot motion, thereby increasing the quality of captured images. This is better than to start with a slow trajectory and gradually increase its speed because the difference in image quality measure (feedback) is more evident in faster movements, which leads to faster adaptation. By stopping the iteration before reaching the desired image quality measure (due to $\gamma < 1$), we ensure that the ILC algorithm does not slow down the motion too much.

2.1.5 Speed profile refinement by reinforcement learning (RL)

Although ILC is able to achieve significant improvements in quite a small number of iterations, its progress stalls once its output approaches the desired one. While this problem

can be addressed with variable gains [51], such an approach often leads to algorithmic instability. In this section, we propose to apply reinforcement learning (RL) to fine-tune the speed profile computed by ILC.

There are many possible RL methods that could be used for this purpose. We selected reward-weighted policy learning with importance sampling (RWPL), which is a simplified variant of Policy Learning by Weighting Exploration with the Returns (PoWER) method [37], for model-free speed profile optimization [68]. It uses a parametrized skill policy and a reward function to maximize the expected return of skill performance trials.

Under the assumption that there is only terminal reward and that only a single basis function is active at any given time (note that this is only approximately true for CDMPs), the control policy parameters $\boldsymbol{\theta}_n$ are updated by RWPL as follows

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \frac{\langle (\boldsymbol{\Theta}_n - \boldsymbol{\theta}_n) R(\boldsymbol{\Theta}_n) \rangle_{w(\boldsymbol{\pi}_k)}}{\langle R(\boldsymbol{\Theta}_n) \rangle_{w(\boldsymbol{\pi}_k)}}, \tag{2.32}$$

where $w(\boldsymbol{\pi}_k)$ denotes the policy parameters of k-th iteration. $\boldsymbol{\Theta}_n = \{\boldsymbol{\theta}_k^*\}_{k=1}^n$ denotes the set of all policy parameters $\boldsymbol{\theta}_k^*$ executed until the n-th iteration and $R > 0$ the terminal reward received at the end of each rollout. $\langle \cdot \rangle_{w(\boldsymbol{\pi}_k)}$ denotes importance sampling [69]. Its role is to select a predefined number of best trials to compute the update in order to reduce the number of iterations until optimal policy is learnt. In the context of speed optimization, parameter $\boldsymbol{\theta}$ corresponds to parameters $\{\boldsymbol{w}_i^\nu\}_{i=1}^M$ that form the temporal scaling factor ν defined in Eq. (2.23).

The update rule specified by Eq. (2.32) is equivalent to

$$\boldsymbol{\theta}_{n+1} = \frac{\sum_{i=1}^m R_{\text{in}(n,i)} \boldsymbol{\theta}_{\text{in}(n,i)}^*}{\sum_{i=1}^m R_{\text{in}(n,i)}}, \tag{2.33}$$

where function $\text{in}(n, i)$ selects the trial with the i-th highest reward from the trial set $\{\boldsymbol{\theta}_k^*, R_k\}_{k=1}^m$, m is the number of best trials selected by importance sampling, and the exploration parameters are computed by adding exploration noise to the current estimate $\boldsymbol{\theta}_n$

$$\boldsymbol{\theta}_n^* = \boldsymbol{\theta}_n + \boldsymbol{\varepsilon}_n. \tag{2.34}$$

Here, $\boldsymbol{\varepsilon}_n$ is zero-mean Gaussian noise. Its variance $\boldsymbol{\Sigma}$ is usually a diagonal matrix and needs to be specified by a user. High variance leads to a more thorough exploration of the parameter space but may cause oscillations around the solution, while small variance can get stuck in a local minimum.

When applying this method to our velocity optimization problem, exploration noise was added to the current weights describing the parametric policy $\boldsymbol{\theta}_n = \boldsymbol{w}_n^\nu = [w_{1,n}^\nu, \ldots, w_{M,n}^\nu]^{\mathrm{T}}$, then robot motion was executed, and the cumulative reward R_n was calculated for the current trial. Importance sampling then selects a subset of best trials to update the weights using Eq. (2.33). We used exploration noise with a relatively small variance because it is safer for the real robot execution and because ILC already created a decent solution. Thus only a small amount of exploration should be sufficient.

Besides selecting the appropriate method, the specification of the reward function is the most important issue in RL. We took inspiration for creating our reward system from the cliff walking problem [70]. Based on this inspiration, at any of the sampled images, our intermediate reward system gives zero reward if the image quality measure is significantly lower than the one learned by ILC. Similarly, the intermediate reward is set to zero if the image quality measure is significantly higher than the reference (desired) value at any of the sampled images because in this case, the inspection process is unnecessarily slowed

down. To compute intermediate rewards, we first define the following values

$$r_{h/v,n,p} = \begin{cases} 0 & \text{if } \phi_{h/v,n,p} \leq \overline{\phi}_{h/v,p} - \Delta\phi_{h/v,p} \\ 0 & \text{if } \phi_{h/v,n,p} \geq \overline{\phi}_{h/v,p} + \Delta\phi_{h/v,p} \\ 2 & \text{if } \widetilde{r}_{h/v,n,p} > 2 \\ \widetilde{r}_{h/v,n,p} & \text{otherwise} \end{cases}, \tag{2.35}$$

where

$$\widetilde{r}_{h/v,n,p} = \frac{1}{\overline{\phi}_{h/v,p} + \Delta\phi_{h/v,p} - \phi_{h/v,n,p}}, \tag{2.36}$$

$$\overline{\phi}_{h/v,p} = (\phi_{h/v,d,p} + \phi_{h/v,\mathrm{ILC},p})/2, \tag{2.37}$$

$$\Delta\phi_{h/v,p} = |\phi_{h/v,d,p} - \phi_{h/v,\mathrm{ILC},p}|/\sqrt{2}. \tag{2.38}$$

In the above equations, index h/v denotes either the horizontal or vertical gradient related values. $\phi_{h,n,p}$ and $\phi_{v,n,p}$ respectively denote the horizontal and vertical image quality measures specified in Eqs. (2.1) and (2.2) and computed at the p-th image captured in the n-th trial. Just like when implementing ILC, these values are aligned along the phase and interpolated to the phases at which the reference image quality measures $\phi_{h,d,p}$ and $\phi_{v,d,p}$ were obtained. Similarly, $\phi_{h,\mathrm{ILC},p}$ and $\phi_{v,\mathrm{ILC},p}$ denote the aligned and interpolated image quality measures obtained when executing the inspection trajectory generated from the parameters computed by ILC.

To favor short execution times, we also define a terminal reward term $\exp(-\kappa T_{e,n})$, where κ is a user-selected positive constant and $T_{e,n}$ denotes the execution time in the n-th iteration. The cumulative reward R_n for the n-th trial is computed by accumulating the intermediate rewards $r_{h,n,p}$ and $r_{v,n,p}$ and adding the terminal reward

$$R_n = \frac{1}{4P}\sum_{p=1}^{P}(r_{h,n,p} + r_{v,n,p}) + \exp(-\kappa T_{e,n}), \tag{2.39}$$

where P is the number of images gathered to compute the reference quality measures (equal to P in Eq. (2.27)). The value of the sum of intermediate rewards and the value of the terminal reward are both between 0 and 1.

Policy learning algorithms other than RWPL could be used for optimizing the speed of motion along the path. However, the number of executions with the real robot is usually a prohibitive factor for methods that require many roll-outs before they update parameters. In this sense, the proposed RWPL method has an advantage over, for example, CMA-ES that requires several executions before an update in order to estimate the natural gradient [53]. On the other hand, among the RL methods, PoWER, RWPL and PI^2 behave similarly in terms of sample efficiency [53]. Section 2.2.4 compares the effectiveness of CMA-ES and RWPL when applied to the problem of speed optimization.

For clarity, a flowchart of the overall approach is provided in Figure 2.5. In this flowchart, ILC-related operations are in the yellow rectangle while RWPL-related operations are in the green rectangle. The optimization process is halted when both ILC and RWPL are completed. The last speed scaling parameters $\boldsymbol{w}^\nu = [w_1^\nu, \ldots, w_M^\nu]^{\mathrm{T}}$ are taken as the final result.

2.2 Experimental Evaluation and Results

In this section, we present several experiments to evaluate the key aspects of the proposed approach: specification and time-parametrization of trajectories for visual inspection and the performance of the speed optimization algorithm.

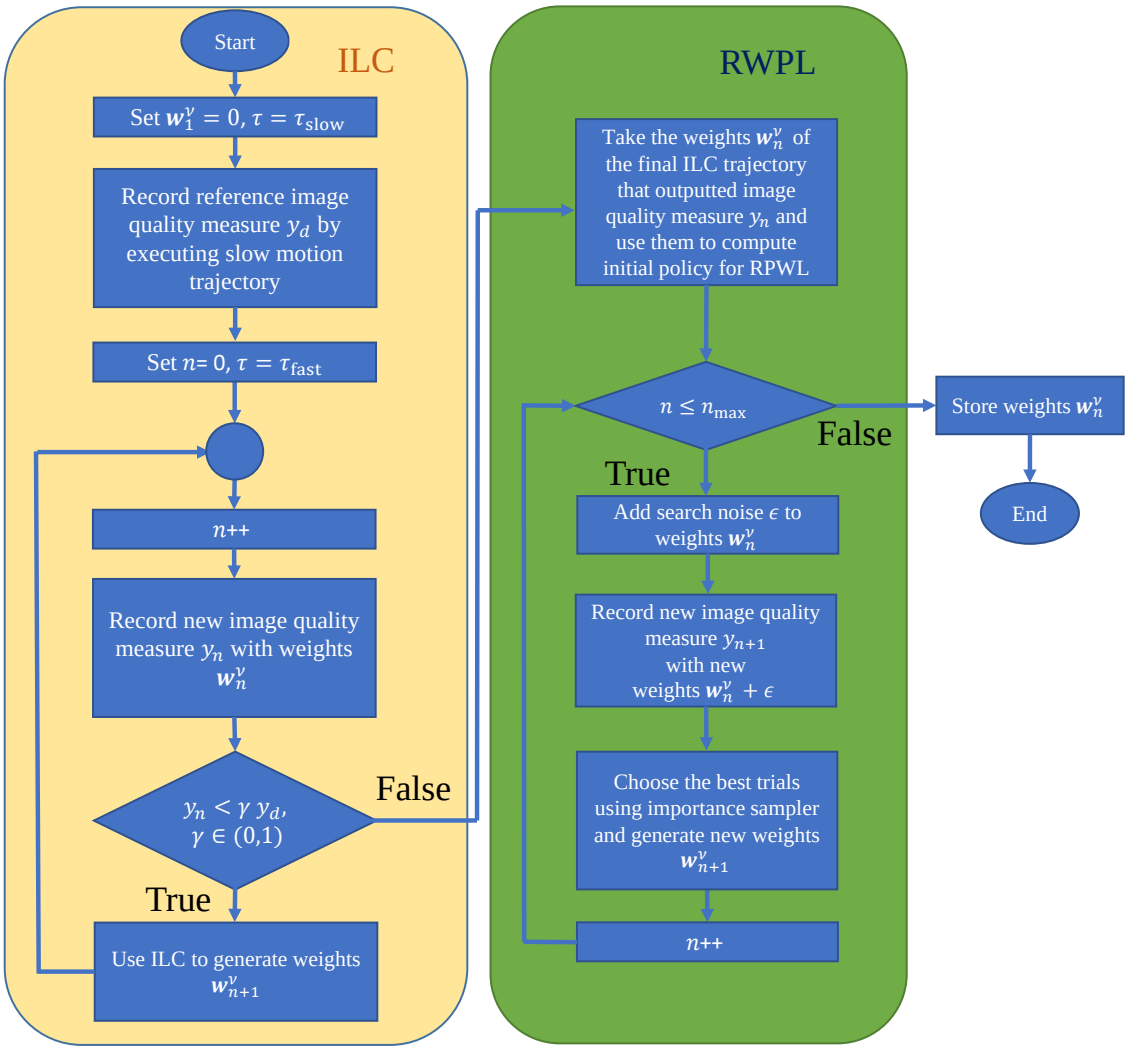


Figure 2.5: Flowchart of the proposed learning algorithm.

2.2.1 Experimental setup

Our experimental setup consisted of the UR-10 robot equipped with an industrial-grade GigE camera – Basler acA1300-60gm with a resolution of 1282x1226 and a maximum frame-rate of 60 fps at full frame (we used 30 fps). As constant lighting conditions are necessary for the image quality measure to be repeatable, we used an additional dedicated LED light. The robot was controlled using a modified version of the UR modern driver [71] available in ROS, which operates in soft real-time at 125 Hz, the maximum control frequency for this robot. The desired position of the robot, obtained by integrating the temporally-scaled CDMP, was updated in every control cycle. Our approach was tested on two different workpieces: the curved workpiece as shown in Figure 2.6-left (here, referred to as *curved* workpiece) and the workpiece with two-sides and a short curvature, shown in Figure 2.6-right (here, referred to as *two-sided* workpiece). Black line on the surface represents the generic feature that needs to be inspected while random text around it represents a random background.

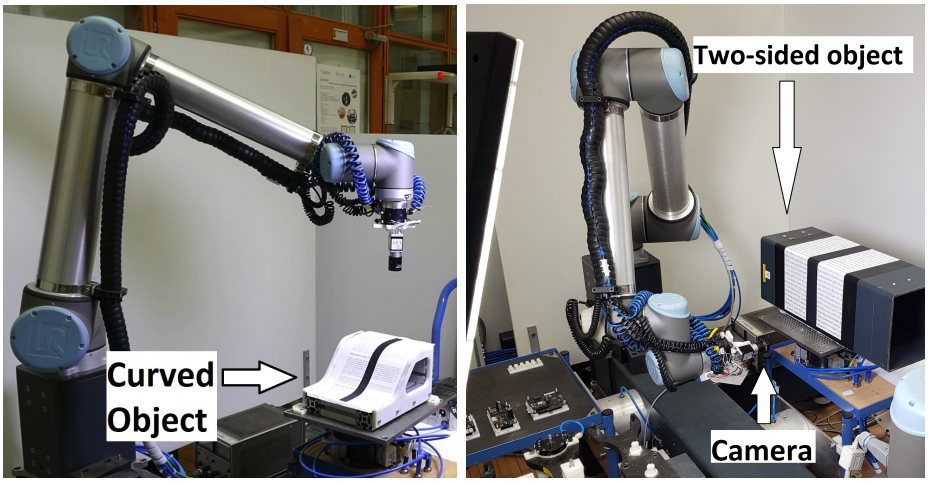


Figure 2.6: Experimental setup with the curved workpiece (left) and the two-sided workpiece (right).

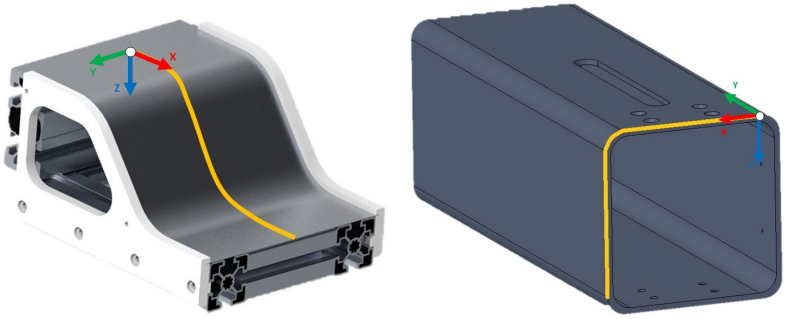


Figure 2.7: The determined paths (yellow), starting points (white) and orientations in CAD models. The curved workpiece is on the left and the two-sided workpiece on the right.

2.2.2 Specification of trajectories

We applied the procedure from Section 2.1.2 to generate the inspection trajectories. The first step in the trajectory generation process was to define the points making up the inspection path, extracting them from the CAD model together with the suitable orientations to define view directions. CAD system PTC Creo [65] was employed for this purpose. In the CAD system, the path designer projected a straight line onto the center of the surface of the curved workpiece and selected the edge of the two-sided workpiece. This led to the paths shown with yellow lines in Figure 2.7. Next, we created the coordinate systems for all points on the desired paths using Eq. (2.5). Since in our experiments, the optical axis of the in-hand camera was aligned with the z axis of the robot, at such orientations the camera was directed towards the workpiece, with the optical axis parallel to the surface normal. The points on each path were sampled so that the Euclidean distances between consecutive points were constant (0.23 mm for the curved workpiece and 0.35 mm for the two-sided workpiece), which resulted in a dense dataset of points. PTC Creo output the sampled positions and orientations on the path as a STEP file. They were then lifted to the desired distance from the two workpieces using Eq. (2.7). For each workpiece, we obtained a sequence of positions and orientations $\{\boldsymbol{p}_k, \boldsymbol{q}_k\}_{k=0}^{K}$.

As shown in Figure 2.8, the resulting position and orientation paths are continuous. To

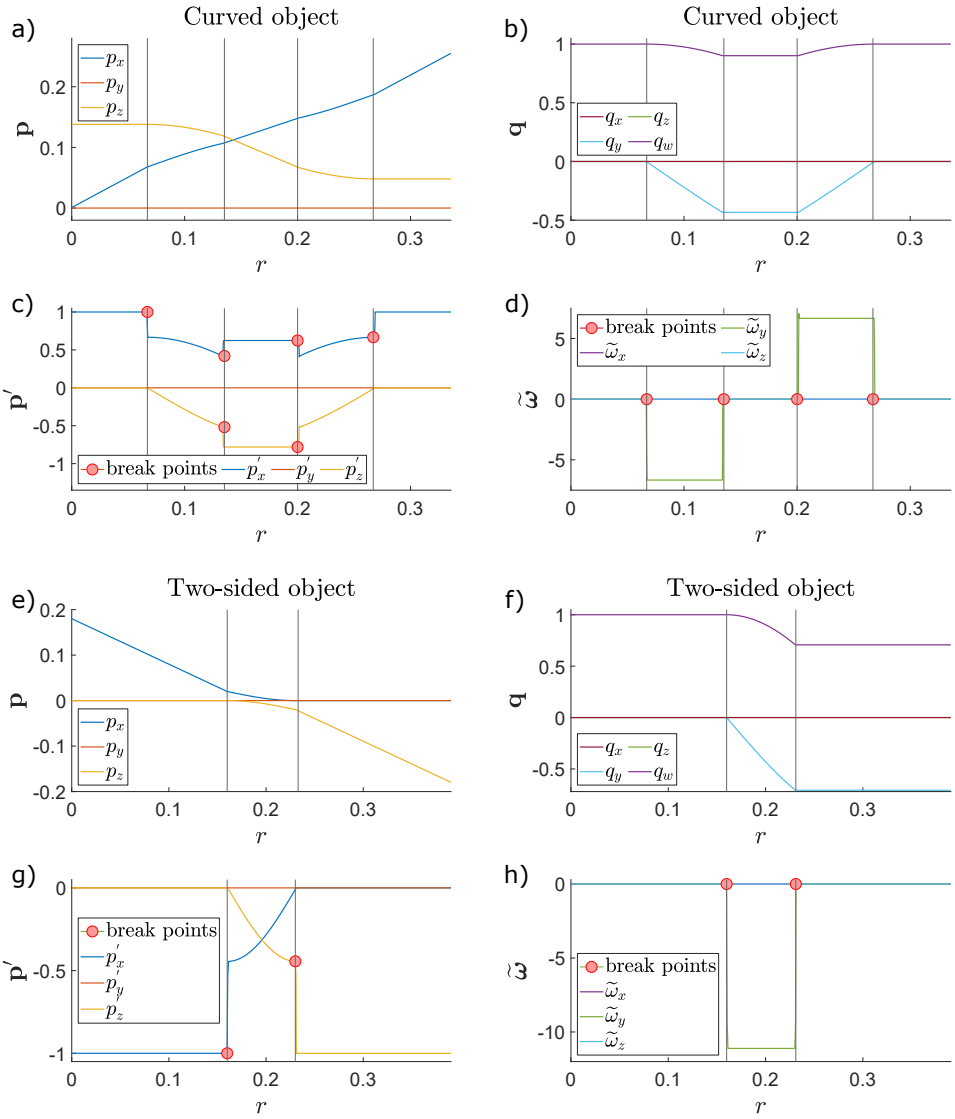


Figure 2.8: Position and orientation paths extracted from the CAD model and their first derivatives with respect to the weighted sum of arc and angle length r as specified in Eq. (2.8). Plots a) b) c) d) show position, orientation, derivative of the position and derivative of orientation with respect to the weighted sum of arc and angle path for the curved workpiece respectively and plots e) f) g) h) show the same parameters for the two-sided workpiece. Discontinuities in the first derivatives are marked with red dots and indicate the breakpoints.

determine if it is necessary to segment the paths, we next computed the linear and angular velocities with respect to the weighted sum of arc and angle length $\{r_k\}_{k=0}^{K}$, $r_k = s_k + \beta\xi_k$, with s_k and ξ_k computed according to Eq. (2.8) and (2.9), respectively. The following formulas were used: $\boldsymbol{p}'_k = (\boldsymbol{p}_{k+1} - \boldsymbol{p}_k)/(r_{k+1} - r_k)$, $\widetilde{\boldsymbol{\omega}}_k = 2\log(\boldsymbol{q}_{k+1} * \bar{\boldsymbol{q}}_k)/(r_{k+1} - r_k)$, $k = 0, \ldots, K-1$, $\boldsymbol{p}_K = \boldsymbol{p}_{K-1}$, $\widetilde{\boldsymbol{\omega}}_K = \widetilde{\boldsymbol{\omega}}_{K-1}$.

The graphs in Figure 2.8 show that the resulting paths are not continuously differentiable. Thus to compute time parametrization, the two paths had to be segmented: into five segments for the curved workpiece and into three segments for the two-sided workpiece. The segmentation points (breakpoints) were determined by the path designer. The method of describing the evolution of time along the inspection path was then used to

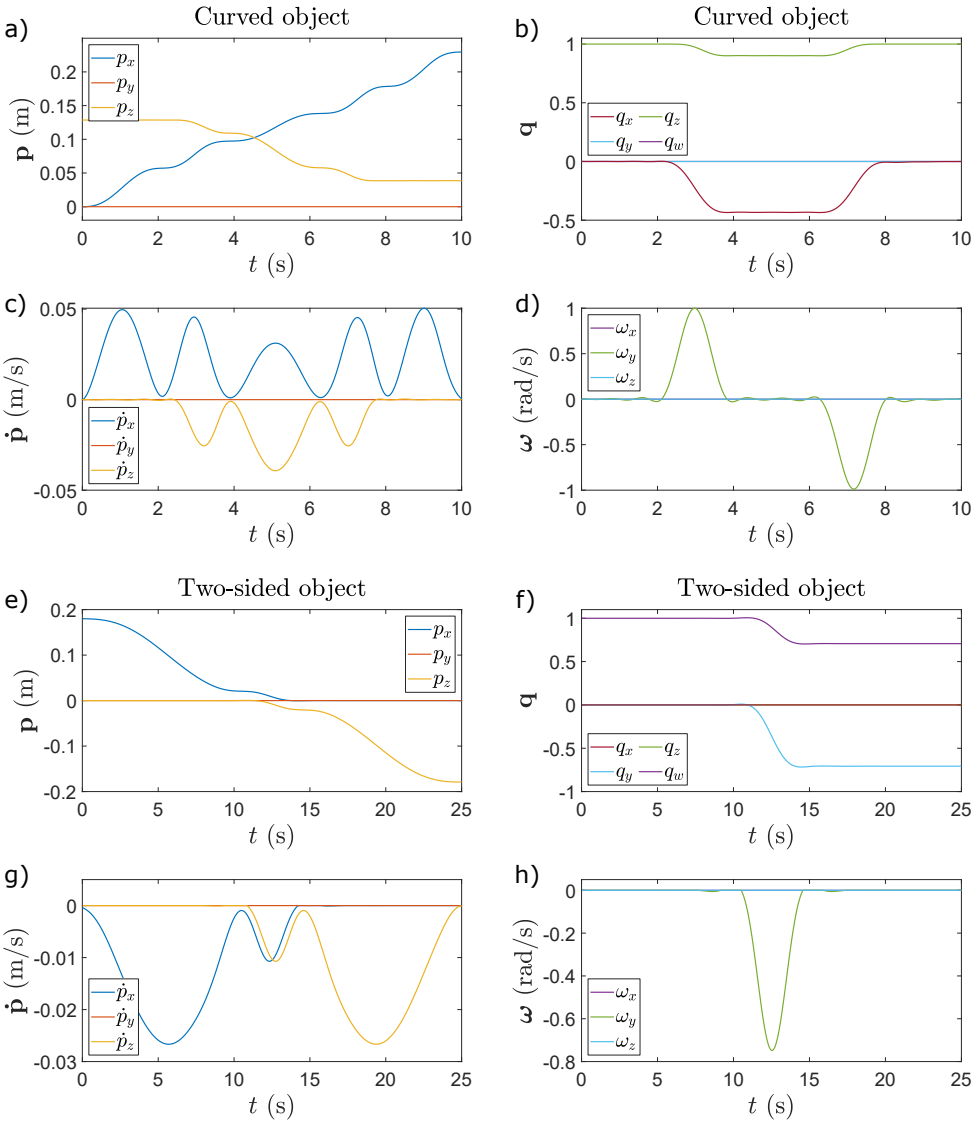


Figure 2.9: Position, orientation, velocity, angular velocity generated with CDMPs for the curved workpiece are given in plots a), b), c), d) and for the two-sided workpiece in figures e), f), g), h).

compute a suitable time parametrization, where the designer set $T_j = 2\,\mathrm{s}$, $j = 1, \ldots, 5$, as the duration for all path segments of the curved workpiece and $T_1 = T_3 = 10\,\mathrm{s}$ and $T_2 = 5\,\mathrm{s}$ as the duration of path segments for the two-sided workpiece.

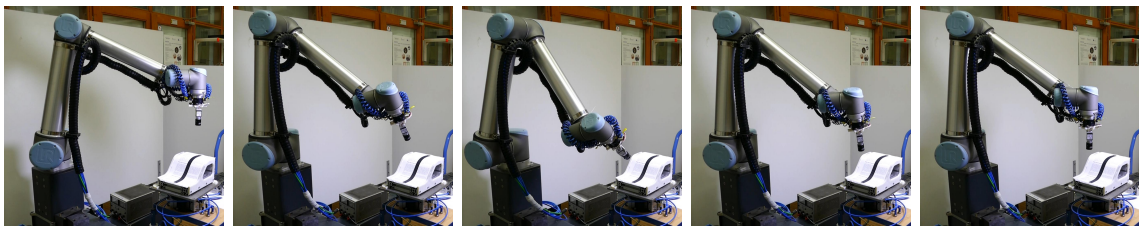


Figure 2.10: Sequence of still images showing the robot moving along the surface of the curved workpiece. Video of the experiments for this workpiece is attached.

As both visual inspections paths are continuous, we were able to encode both of them as one speed-scaled CDMP [48]. We used $N = 25$ weights to encode the desired trajectories with CDMPs and $M = 25$ weights to encode the velocity profiles within CDMPs. The resulting trajectories and their velocities are shown in Figure 2.9. As the duration of the trajectory was set to $T_{e,\text{fast}} = 10\,\text{s}$ for the curved workpiece and $T_{e,\text{fast}} = 25\,\text{s}$ for the two-sided workpiece, we set $\tau_{\text{fast}}^{\text{curved}} = 10$ and $\tau_{\text{fast}}^{\text{two-sided}} = 25$. The execution of both trajectories at these speeds resulted in blurred images. The reference image quality measures $\{y_d(x_p)\}_{p=1}^{P}$ were then gathered by executing both CDMPs with $\tau_{slow} = 70$.

2.2.3 Performance of speed optimization

Next, we applied the learning process to both visual inspection trajectories as presented in Figure 2.5. We ran ILC for maximum of 20 iterations and stopped it when 95 % of the desired image quality measure was reached, i. e., $\gamma = 0.95$ in Eq. (2.31). After that, we used the ILC-learned trajectory as the initial trajectory for RWPL, which was used to further refine both trajectories. Figure 2.10 shows one iteration of learning performed on the curved workpiece.

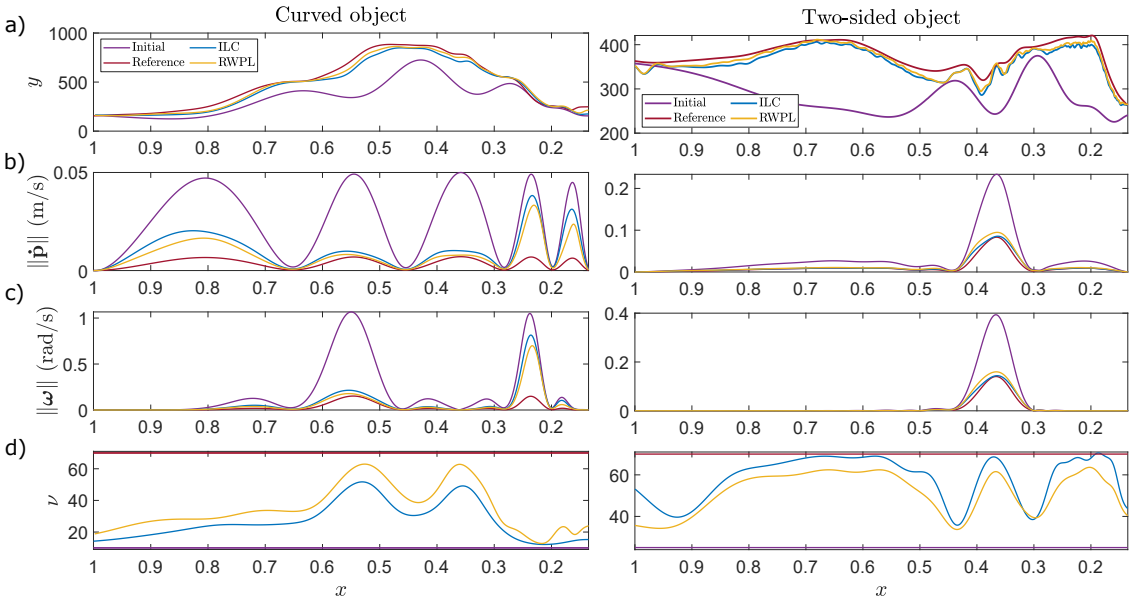


Figure 2.11: Optimization results for the curved workpiece (left column) and the two-sided workpiece (right column). The graphs in a) show the image quality measures y. The next three lines of graphs show the linear and angular speed $\|\dot{\boldsymbol{p}}\|$ and $\|\boldsymbol{\omega}\|$, in rows b) and c) respectively, and the corresponding speed profile ν in row d). The graphs show these quantities for the initial motion (too fast), reference motion (too slow), and the motions learned by ILC and RWPL.

The graphs in the left column of Figure 2.11 show the results of the optimization process for the curved workpiece. To improve the quality of images, ILC adapted the speed scaling factor so that the trajectory duration was modified from the initial 10 s (purple line), which resulted in blurred images, to 25.7 s (blue line). RWPL further slowed down the motion to 32.3 s (yellow line). This way we got closer to the desired image quality measure y_d obtained with the reference, 70 s trajectory (red line). The right column of Figure 2.11 shows the results of the same process for the two-sided workpiece. In this case, ILC learning started at 25 s and adapted the speed scaling factor so that the image quality measure reached 95 %

of the desired measure, which resulted in 55.1 s long trajectory. Finally, RWPL changed the duration of motion to 52.3 s.

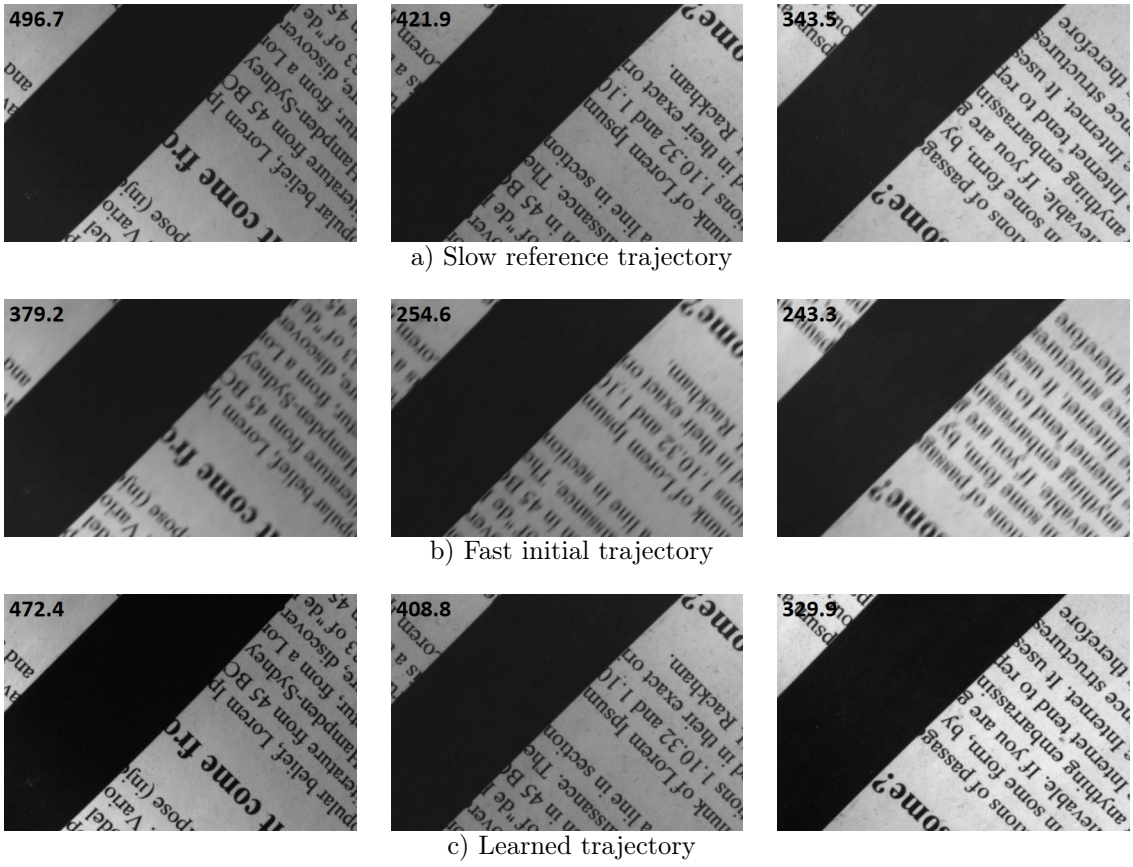


Figure 2.12: Effect of motion speed on the captured images. Row a) shows the images for the slow reference trajectory, where image quality is high but the motion is slower than necessary. Row b) shows the images for the case of the initial trajectory, which is too fast to obtain high-quality images. Row c) shows the images obtained while executing the optimized trajectory after applying ILC and RWPL. Column-wise, the images were taken at the same phase of the motion in all three instances. The current image quality measure value is displayed in the upper-left corners of the images (values for the experiment shown in the attached video).

With ILC we obtained 95 % of the desired image quality measures in 63.3 % less time for the curved workpiece and 21.3 % less time for the two-sided workpiece. With RWPL, we achieved further improvements in 100 iterations, leading to 98.9 % of the reference image quality measure in 53.9 % less time than the reference trajectory for the curved workpiece. For the two-sided workpiece, RWPL reached 97.1 % of the desired image quality measure while also decreasing ILC time (from 55.1 s to 52.3 s), resulting in 25.3 % overall time reduction from the reference trajectory. Some of the images gathered while performing visual inspection movements, with the initial speed and optimized speeds after learning, are for qualitative comparison shown in Figure 2.12. The current image quality measure value, displayed in the upper-left corner of each image, quantitatively confirms that the image quality measure computed from images acquired with the hand-held camera while the robot moves along the optimized trajectory is very close to the reference trajectory, even though the optimized trajectory is much faster.

2.2.4 Comparison to other methods

As stated previously, other methods than the combination of ILC and RWPL can be used for the black-box optimization. We compared the proposed approach with CMA-ES, which is considered state of the art in stochastic optimization [22], [53]. Unlike RWPL, it requires several roll-outs to estimate the natural gradient.

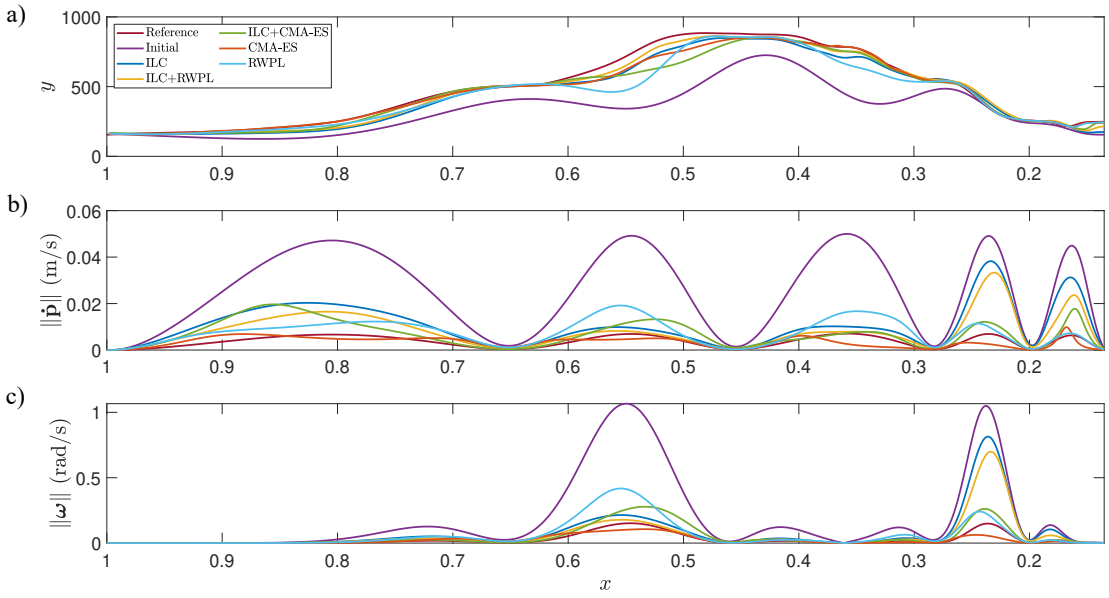


Figure 2.13: Comparison between initial, reference, ILC, ILC and RWPL, ILC and CMA-ES, only CMA-ES and only RWPL for a) image quality measures (y), b) linear speed $\|\dot{\boldsymbol{p}}\|$ and c) angular speed $\|\boldsymbol{\omega}\|$.

Figure 2.13 shows the image quality measure and speeds for different situations. Note that the resulting image quality measure is qualitatively the same when the motion is optimized by either the combination of ILC and RWPL, ILC and CMA-ES, or by only CMA-ES and RWPL. The same reward function, defined by Eq. (2.39), was used by all algorithms. Note that twice as many real-world executions were performed for CMA-ES only and RWPL only (240, while ILC and RL performed 20+100 trials). However, the results in Figure 2.14 show that the execution times achieved by combining ILC and CMA-ES or by only CMA-ES or RWPL are considerably longer than the execution time achieved by the proposed ILC and RWPL combination. While ILC only achieves a shorter execution time, the computed image quality measure is worse because ILC cannot ensure convergence to the desired image quality measure.

These results confirm that different methods can be applied to minimize the execution time without degrading image quality, but the complexity of the methodology and the final result might not always be the same. Either different rewards might be needed, or more parameters might have to be set with different approaches. For example, for the CMA-ES, we needed to determine the number of roll-outs before an update. In this case, we used empirically determined 4 roll-outs. A higher number would result in a linearly higher number of repetitions, but not a linearly better result.

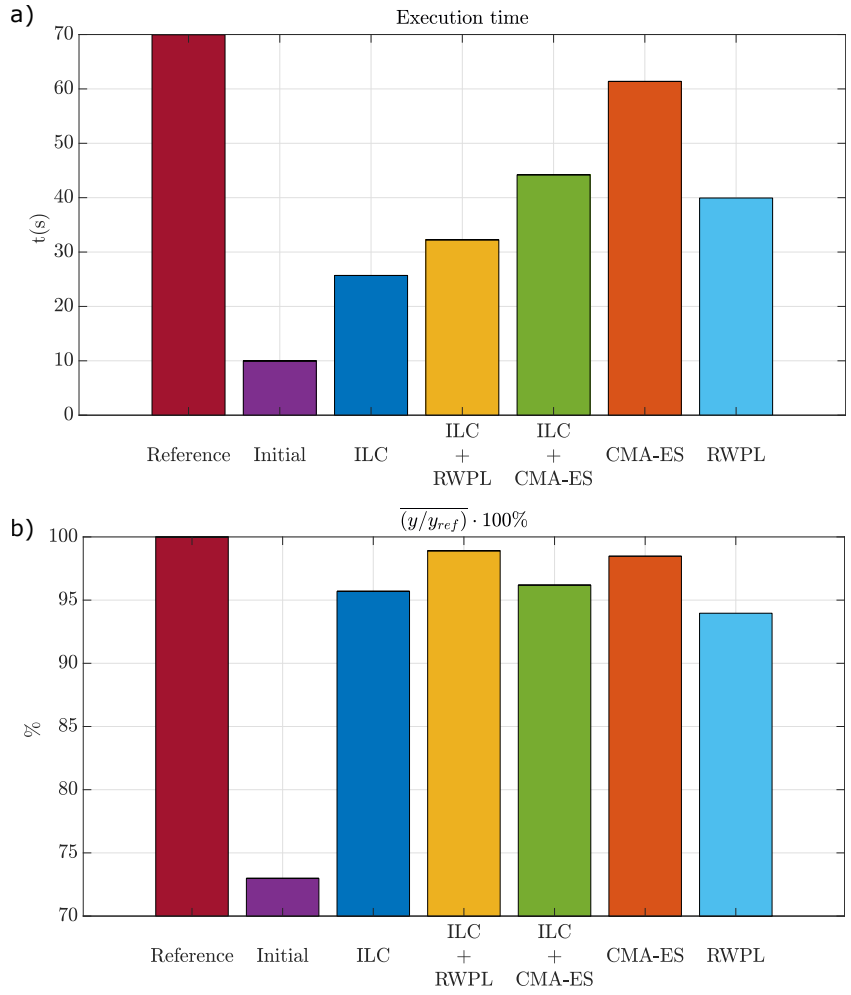


Figure 2.14: Comparison between initial, reference, ILC, ILC and RWPL, ILC and CMA-ES, only CMA-ES and only RWPL for a) execution times and b) average image quality.

2.3 Discussion

One question that arises is if we could calculate the desired speed without applying the presented optimization process, or any other optimization method. Let us assume that the camera is perpendicular to the surface of the workpiece and that it moves in x direction. The following expression can then be derived to relate camera speed to image motion [72]

$$\dot{p}_x = \frac{d_f \kappa u_x}{t_e f}. \tag{2.40}$$

Here, d_f is the distance of the camera from the workpiece, f is the focal length of the lens, t_e is the camera exposure time, u_x is the pixel size, and κu_x is the image motion that occurs while the iris is open. In our experimental setup, we had $d_f = 0.366\,\mathrm{m}$, $t_e = 0.0115\,\mathrm{s}$, $f = 25\,\mathrm{mm}$, $u_x = 5.61\,\mu\mathrm{m}$, thus to keep the motion blur below 1 pixel, i. e. $\kappa = 1$, the camera speed should be equal to

$$|\dot{p}_x| = \frac{0.366\,\mathrm{m} \cdot 1 \cdot 5.61 \cdot 10^{-6}\,\mathrm{m}}{11500 \cdot 10^{-6}\,\mathrm{s} \cdot 25 \cdot 10^{-3}\,\mathrm{m}} = 0.0071\,\mathrm{m/s} \tag{2.41}$$

or less. Figure 2.11 shows that for slow motion, which was used to collect $y_d(s_j)$ ($\nu_{\mathrm{slow}} = 70$), the maximum linear speed on different segments was between $0.007\,\mathrm{m/s}$ and $0.009\,\mathrm{m/s}$.

This is close to the computed maximum speed of 0.0071 m/s, which guarantees that image blur is below 1 pixel. Thus, the initial, empirically determined speed was appropriate and did not introduce significant motion blur.

It would be possible to optimize the speed of motion by specifying a nonlinear constrained optimization problem, where one of the constraints would be the maximum allowed speed. In this case, the optimization could be performed off-line and compute the fastest motion trajectory that does not exceed the maximum allowed speed. However, there are many disadvantages of such an approach. Firstly, it assumes that the camera parameters are known with high accuracy, which requires an accurate calibration process, which is otherwise often not needed for quality inspection. Even small calibration errors can introduce a substantial error in the computed maximum speed. Secondly, such an approach cannot take into account other factors that affect image quality, e. g. camera vibrations introduced by robot motion, errors in the specified workpiece-to-camera distance that might arise during motion, lighting conditions, robot dynamics and other sources of noise. On the other hand, the proposed approach encompasses all of these aspects in learning and thus greatly simplifies the complete trajectory specification process. Since it optimizes an objective measure for image quality, it considers all aspects that affect image quality and does not require the user to set an arbitrary parameter to limit motion blur. While motion blur is obviously an important factor to determine image quality, it is not the only one.

The above-mentioned factors are also detrimental when generalizing between objects by utilizing the learned speed profile in one setting for another object/setting. While this kind of generalization could theoretically provide an initial policy for learning provided there are the same lighting conditions, robot, and camera in the workcell, this could prove more problematic and costly than just learning again from scratch. Different object geometries namely often require different speed profiles.

The optimization problem (2.24) is highly nonlinear, therefore we cannot theoretically prove that the proposed method will find a global optimum. The proposed approach is based on policy search methods that only consider the current policy and its neighborhood in order to gradually improve the performance. This might result in a local optima [38]. However, because the selected image quality measure monotonically strictly decreases with the increased velocity, the proposed algorithm is able to approach the locally optimal solution. Moreover, we have shown in our experiments that the proposed method significantly reduces the time of execution while maintaining the initial image quality measure.

Our experiments made clear that even with fairly benign workpieces like the ones shown in Figure 2.7, which involve a combination of flat and curved surfaces, discontinuities in first derivatives can arise. The only way to accurately follow such trajectories with a real robot is to reduce the velocities to zero at breakpoints. Our approach to trajectory specification is mathematically rigorous and ensures that velocities are set to zero where needed so that the robot can instantaneously change the direction of motion at breakpoints. This way we compute physically feasible trajectories. Otherwise, inaccuracies in tracking would inevitably occur, which is inappropriate for visual quality inspection tasks. Another benefit of the proposed approach is that the speed is adapted by multiplying the initial velocities by a scaling factor, which is learned on-line. The learning procedure is therefore guaranteed to preserve the zero speed at breakpoints, thus ensuring that the optimized trajectories are also feasible.

Our approach is to keep the user in the loop where she or he is needed, i. e. when specifying the course of motion along the workpiece surface. To specify an appropriate path, the user must understand the desired visual inspection task in order to specify a path on the surface of the workpiece's CAD model and determine the breakpoints to segment the paths into twice continuously differentiable parts. All other parts of the proposed

approach are fully automated and rely on an objective measure of image quality that does not require the user to subjectively assess the quality of images. In addition, the proposed methodology does not require any knowledge about camera parameters.

A considerable factor in manufacturing is the amount of personnel time required to setup a new production cell. In our experience, it usually takes a long time to manually fine-tune the speed of motion. In contrast to this, in the proposed approach, the user simply starts the learning process and comes back when it is finished. Thus, the personnel costs can be reduced since the user has time to perform other tasks while the optimization process is running.

In this chapter, we developed a new methodology for optimization of velocity for the robot trajectories. Although optimization itself is the main topic, our approach required also complicated parametrization of the initial robot trajectory, where we needed to extract the path from the CAD model of an object and to calculate the time parametrization. In order for systems in the factories of the future to be able to do this process autonomously, we need to go to the lower levels of our pyramid (Figure 1.4) that are able to generate and adapt motions of the robots.

Chapter 3

Adaptive Process Behaviour

In the preceding chapter, we presented an instance of optimizing a pre-existing trajectory. Nonetheless, there are other means to accelerate the entire process. Thus, in this chapter, we present an operational example of process adaptation. In this case, we know what is an overall goal of our task, we have a certain knowledge about it, and we want to make it faster. This task is considered more intricate (located lower in the pyramid - Figure 3.1) due to the absence of prior information on the parameters used to optimize the task and the trajectories for its accomplishment. To make decisions regarding the system behaviour, given the multitude of parameters involved, we advocate for the adoption of an NN classification technique. To tackle the issue of trajectory generation, we propose using motion planners. Although there are numerous motion planners already in use, we provide a brief overview of the basic ones, without delving into their implementation details. The implementation details are presented in [58], and we outline a methodology for selecting the most appropriate one for the specific task.

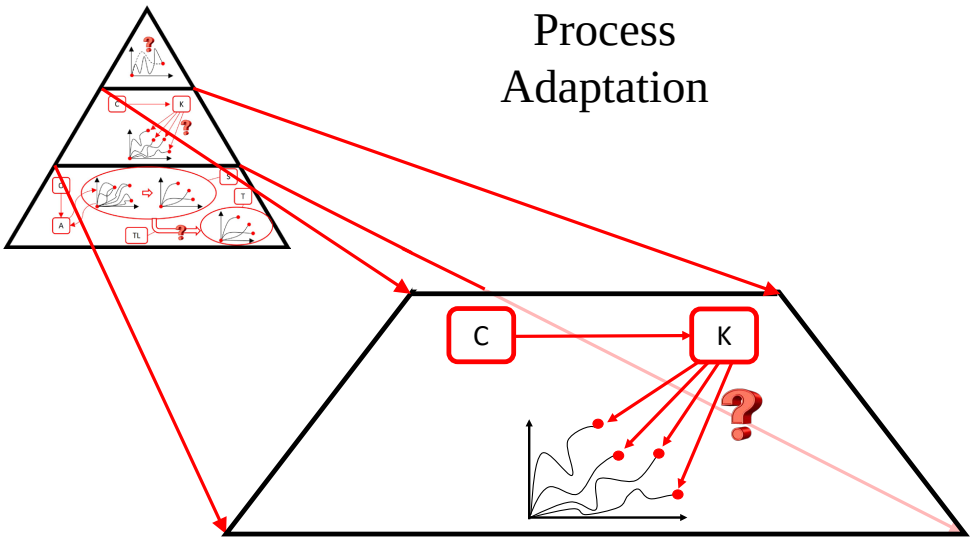


Figure 3.1: Trajectory adaptation is set to be in the middle of our classification pyramid since it is occasionally used in industrial processes but still not completely generative behaviour approach.

We demonstrate the advantages of our approach on the example of industrial line that produces parts for automotive industry through injection moulding. These parts

have several possible defects that are the most common to appear. Vision techniques for checking the products already exist [73], [74], but the problem for our case is that there are too many possible defects, and checking all of them with a robot would create a bottleneck in the production lines. Our approach enables the elimination of improbable defects and rapid generation of the trajectories for the transition between inspection points of the probable defects.

This study concentrates primarily on two topics: evaluating motion planners and identifying the part of the product that requires inspection based on the process parameters. To determine the required viewpoints for inspecting the product, we utilize an NN that takes the process parameters as inputs and produces the potential defects as outputs (C in Figure 3.1). Since the environment is always stationary and known, we have some knowledge about the task (K in Figure 3.1). This includes a model of a cell and a robot (we know where are the obstacles) and the desired positions of the camera in relation to the object for each of the possible defects. With this knowledge, we employ motion planners to navigate an in-hand camera around the static object and position it appropriately, ensuring that there are no obstructions in the line of sight [17]. Motion planners are a suitable solution for this task since they can generate movements using only start and goal configurations, rather than a complete specification of intermediate states [75].

3.1 Methodology

Our method relies on the capability of NN to extract dependency of possible defects in products and the process parameters and capability of motion planners to generate fast trajectories in confined spaces in real-time. Different motion planners exist, based on different approaches and with different results of the application [19]. For the completeness of the thesis, we introduce the most generic ones in Section 3.1.1. We continue with Section 3.1.2 where we show how we can experimentally determine the correct motion planner for the given use-case through an in-depth statistical analysis of a set of readily available motion planners and tools [59]. In order to determine which discrete point needs to be checked (where the error is most likely to occur), we need to follow the process parameters. In Section 3.1.3, we introduce high-level decision-making technique that is based on NNs that follow the aforementioned parameters and output the label of the viewpoint that needs to be inspected (C in Figure 3.1).

3.1.1 Planners

The task of path-planner is to "find a collision-free motion between an initial (start) and a final configuration (goal) within a specified environment" [21] as shown in Figure 3.2.

Figure 3.2: Path planner inputs and outputs.

Characterization of a motion planning problem can be done according to the task it addresses, properties of the robot and properties of the algorithm [75].

By the task it solves, motion planning can be divided into four categories: navigation, coverage, localization and mapping. Navigation refers to finding a path that connects initial and goal position. Coverage is planning of the movement that will cover a certain amount of points in space. Localization is the problem of using a map to interpret sensor data to determine configuration of the robot, while mapping is exploring of environment in order to construct a representation useful for all other tasks.

Motion planning heavily depends on DoF of the task space and configuration space of the robot. When configuration space is defined, we have to be aware of constraints (joint limits) of the robot and if the robot is modelled using kinematic or dynamic equations in order to know if the output of the planner needs to be torques (forces) or angles (offsets) of joints.

Properties related to the algorithms refer to optimality of the motion, computational complexity, completeness and finding solutions online or offline. Motion can be optimal in terms of time, energy, or path length. With changing the size of a problem description (number of DoFs, number of obstacles), memory requirements and running time consequently change. These changes refer to the complexity of the algorithm. Planners can be complete (solution is found if it exists, otherwise failure is indicated), but for the high dimensionality of the problem (usually number of DoFs), these algorithms become computationally too expensive. That is why weaker forms of completeness are introduced: resolution completeness (if solution exists on the given level of discretization, it will be found) and probabilistic completeness (probability of finding solution converges to 1 as the time goes to infinity). Division of online/offline algorithms tells us if the solution is found based on model and then sent for execution (offline) or it is generated while executing (online).

The robot path planning problem in literature is usually divided into classical methods and heuristic methods, as shown in Figure 3.3.

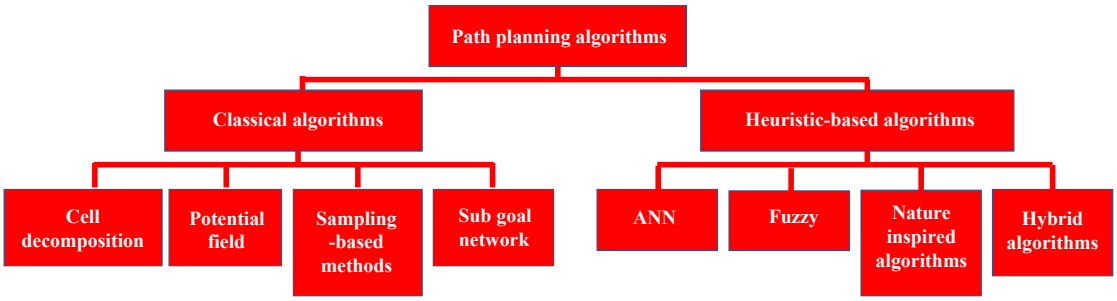


Figure 3.3: Path planner division introduced in [19].

In the following subsections we give a brief explanation of all the methods as well as benefits and drawbacks of using them. For graphs and details about implementation, see [76].

Classical Algorithms

Classical algorithms are very popular in solving robot motion planning tasks because it is observed that either a result would be obtained, or it would be confirmed that a result does not exist [77]. The major drawback of this approach is high computational cost and failure to respond to the uncertainties of the environment, so it is less preferred for real-time implementation. This is why for motions in dynamic environments, where objects are not stationary, heuristic methods based on artificial intelligence (AI) are becoming more

popular. In the following subsections, we briefly review several classical approaches: Cell Decomposition (CD), Potential Field, sampling based methods (Probabilistic Road-Map and Rapidly-Exploring Random Trees) and Subgoal Networks.

In the Cell Decomposition (CD) method, robot's workspace is divided in regions called cells (Figure 3.4-left). Based on this division, adjacency graph is formed (Figure 3.4-right). In this graph, each node corresponds to one cell and edges correspond to the connection between adjacent cells. Path planner first determines cells with start and goal point and then searches for a path on the graph. Mainly used algorithms for this purpose are Dijkstra's, A-star and Sample algorithm.

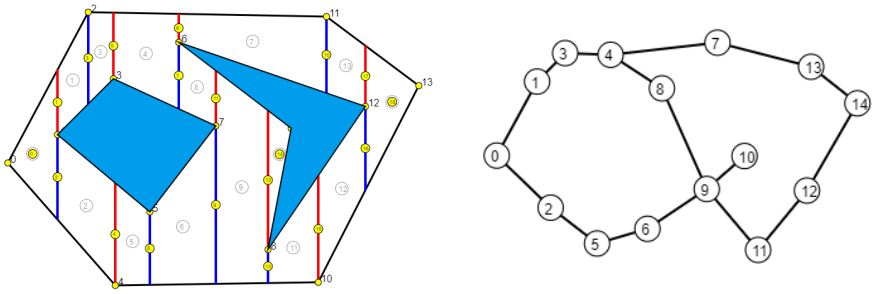


Figure 3.4: Workspace divided to cells (left) and corresponding graph (right) [78].

The basic concept of Potential Field method is to fill the robot's workspace with a potential field, in which the robot is attracted towards the target and repulsed from the obstacles. At each point, the robot heads in the direction that is negative of the gradient of potential field. Figure 3.5 shows an example of robot navigating through potential field from starting (red dot) to goal position (green dot) [79]. Obstacles, presented with the black areas, generate repulsive force and goal position generates attractive force. Resulting vectors are presented with arrows.

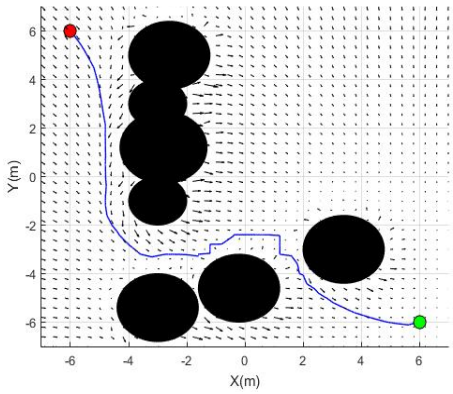


Figure 3.5: Robot navigating through potential field from starting (red dot) to goal position (green dot). Obstacles, presented with black areas, generate repulsive force and goal position generates attractive force. Resulting vectors are presented with arrows. Reprinted from [79], used under Creative Commons CC-BY license.

Sampling-based methods for planning (SBP) have been widely used recently because of their capability to solve time-critical real-world planning problems. They are unique by the fact that planning is done in the configuration space of the robot. SBP is trying to capture the connectivity between randomly sampled points. The most commonly used

algorithms of this subgroup are Probabilistic Road-Map (PRM) [80] and Rapidly-Exploring Random Trees (RRT) [81]. Both algorithms are based on the same idea of connecting randomly sampled points, but different in the manner they construct a graph connecting the points [82]. The main advantage of these two methods is that they are capable of finding fast solutions for difficult problems, and the disadvantage is that they have a weaker notion of completeness (probabilistic completeness) and that solutions are widely regarded as suboptimal. Probabilistic Road-map (PRM) algorithm consists of two stages: Graph building stage and planning stage. In the graph building stage, random points are sampled in the configuration space. Points that are in collision are rejected (Figure 3.6-left). The algorithm tries to connect all pairs of the vertices with edges (curves or straight lines) (Figure 3.6-center). In the planning stage, the graph developed in the previous stage is used and the shortest path is found (Figure 3.6-right). Any graph searching algorithm can be used in this stage (A-star, Dijkstra's).

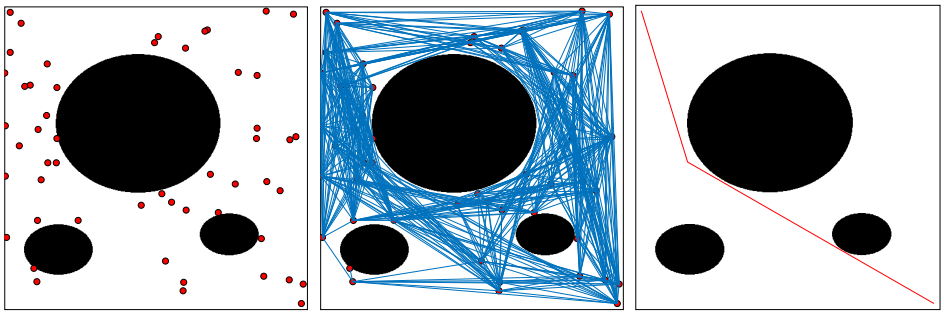


Figure 3.6: PRM steps: 1. Sampling random points (left), 2. Searching for possible connection between points (middle), 3. Finding the shortest path from the upper left corner to the bottom right corner [76].

Rapidly-exploring Random Trees (RRT) algorithm grows the tree so that each node of the tree is a state in the workspace (Figure 3.7). The algorithm starts at the initial position of the robot. Before a new state is added to the tree, a random point is generated in the workspace. The algorithm then checks if there is a connection between the closest node in the tree and this random point. If there is a connection, the tree is expanded towards the random point by a small step. If there is no connection, a new random point is generated. This is repeated until goal state is reached. In RRT, step size is the only parameter. Smaller values of this parameter result in slow expansion, but finer paths. The expansion of the tree can be biased towards the goal by choosing the goal state as the random state with the certain probability.

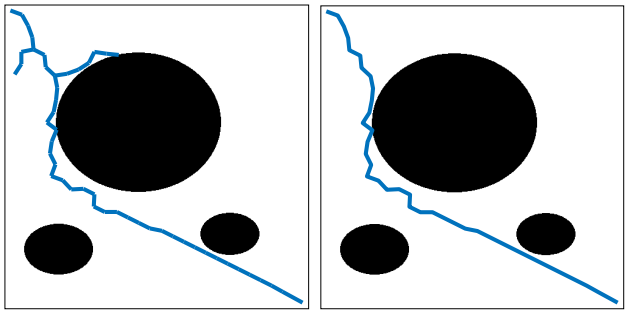


Figure 3.7: RRT tree building (left) and final path (right) [76].

The subgoal network builds the list of reachable configurations from the initial configuration instead of building the representation of obstacles. The connection between two configurations is decided by a local operator motion planning algorithm. This algorithm connects two configurations using a straight line. First, candidate sequence of subgoals is formed. If it does not reach the goal, reachable subgoals are stored, and it tries to find the sequence of configurations between the goal and any of the subgoals. The main advantage of this approach is that it has small memory requirements.

Heuristic Algorithms

Classical approaches have been dominant in robotics for years, but recently heuristic algorithms are becoming more and more popular. The main reason for this is that classical approaches do not produce optimal paths and tend to be locked in some local minima. Moreover, some of them may not provide the suitable solution in the presence of multiple obstacles or in dynamic environments [19]. In the following subsections, some of the most widely used heuristic approaches are presented.

Expansion of NNs and their properties such as nonlinear mapping, good generalization performance and ability to learn from examples have led to them being employed in navigation task for modelling complex relationships between inputs and outputs. Robot navigation problems solved by neural networks can be categorized in interpreting the sensory data, obstacle avoidance and path planning [83]. Sensors are necessary for a robot to determine its position in space. They can be roughly divided in two classes: internal (accelerometers, gyroscope, encoders) and external (lasers, infrared sensors, sonar, visual sensors). Internal sensors are prone to accumulating error and external can have a lot of noise. Since neural networks have many processing nodes, they are able to provide robustness and fault tolerance for interpretation of sensor data. Feedforward networks and Principal Component Analysis (PCA) are heavily used in correcting the data from sensors [84], translating readings from sensors into occupancy values of each grid cell for building metric maps, combining information from two different types of sensors [85], compressing dimension of sensor data or extracting important features [86]. Neural networks for obstacle avoidance are taking the data from the environment as inputs and they output the direction in which the robot should move. Mainly used networks for this purpose are multilayer-perceptron and (MLP) [87] and recurrent neural networks (RNN) [88]. These networks are often combined with networks used for interpretation of sensory data [89]. Classical methods suffer from the problem of being exponentially more demanding as the dimensionality of the problem increases. On the other hand, if all the executable paths in some environment would be stored, it would require an enormous amount of memory. Neural network path planning solutions are based on examples provided by human participants [90] leading the robot in known environment from randomly chosen start and goal position or by randomly generated feasible paths by some of the classical algorithms [91]. For all the networks, input data should consider robot's current and previous state, goal position and sensory data so it is able to output commands for changing the state. The main drawback of neural network approach is that results cannot be explicitly explained and that learning is time consuming. The benefit of this approach is that once the network is trained, the time of execution is not dependent on the size of the problem anymore and that is why it does not have a trade-off between speed and optimality as classical approaches.

The term fuzzy refers to things that are not clear. It offers flexibility in reasoning when considering uncertainties in any situation. Path planning using fuzzy logic is considered reactive planning because the path is planned only for immediate move without caring about the future. In order to design path planner that uses fuzzy logic, we need to formulate

inputs and to create design rules (form of weighted IF-THEN) that will provide one of the formulated outputs to be active. In the example shown in Figure 3.8, inputs to the planner are distances from the obstacle in front, front left diagonal, front right diagonal, angle between the heading direction of the robot and the goal, distance from the goal and preferred turn. Based on these six inputs, the robot decides whether to turn clockwise or anti-clockwise. This way, logic is written so that the robot avoids obstacles and aligns itself towards the goal. The major drawback of this approach is that it requires a lot of tuning of the rules, and it requires all the possible rules to be considered by the programmer.

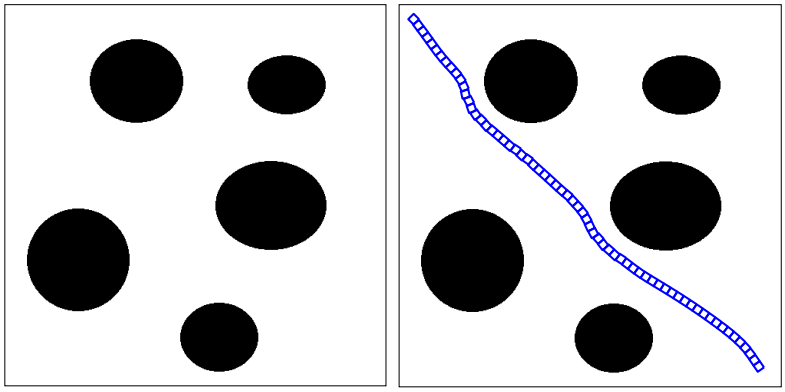


Figure 3.8: Finding path from the upper left to the bottom right corner with fuzzy logic [76].

Nature-inspired algorithms which are robot navigation techniques that are inspired by biology behaviours have gotten much attention recently. The ones that demonstrated the best performances are: Genetic Algorithms (GA), Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO).

A genetic algorithm (GA) is a search heuristic inspired by Charles Darwin's theory of natural evolution. This algorithm simulates natural selection so that the fittest individuals are selected for reproduction to produce offspring for the new generation. In genetic algorithm, five phases are considered: 1. Initial population, 2. Fitness function, 3. Selection, 4. Crossover, and 5. Mutation. In the first step, the population is chosen randomly. For the case of path finding, the population contains all the paths that are feasible for the execution. These paths are usually found by connecting start and goal position using greedy approach based on Euclidean distance heuristic. Each member of the population is defined as a chromosome, and each chromosome is defined by the set of genes. In the use-case of path planning, this means that each feasible trajectory is one chromosome. This means that the number of break points (turns) in the trajectory is a parameter that is given by the programmer. Genes (encoded in binary notation) represent coordinates of the path points. This means that these points coordinates are our optimization variables. As the objective (fitness) function, the length of the path is used in order to optimize the path. This refers to the second and third phase, where only the best chromosomes are chosen. The crossover is done between the selected ones. It means that the algorithm takes some genes from the best chromosomes and creates the new offsprings (paths). With certain probability and in order to maintain diversity within population and premature convergence, mutation phase is executed over some of the offsprings. This means that some genes in the chromosomes obtained by offspring can be flipped. The algorithm terminates when the population has converged (there is no significant difference between successive generations). In Figure 3.9-left, average fitness value, distance between individuals within generation and the path length through 10 generations are given. The path found by the algorithm is given in

Figure 3.9-right. The number of break points used was 2, meaning that each chromosome had a number of genes that corresponds to representing coordinates of 2 points in binary notation.

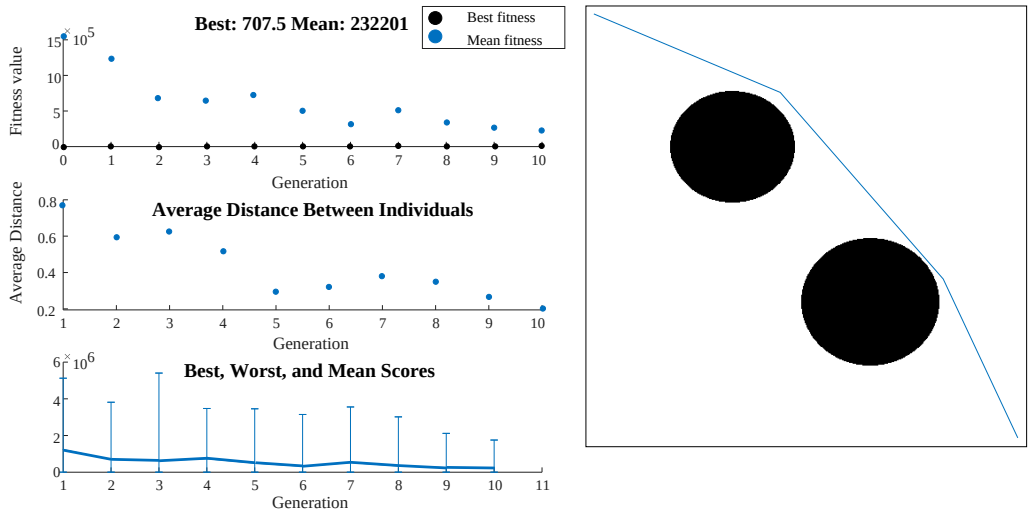


Figure 3.9: GA fitness, distance within the population and path length through iterations (left) and solution of the path from the upper left to the bottom right corner (right) [76].

PSO is mainly inspired by social behaviour patterns of organisms that live and interact within large groups. It incorporates swarming behaviours observed in flocks of birds, schools of fish or swarms of bees [92]. It is a stochastic global optimization method based on population that is especially useful for parameter optimization in continuous multidimensional search spaces. Similar as in GA, PSO creates a random population (in our case feasible paths) and uses objective function (path length measurement) to evaluate particles. However, mutation and crossover operations disappear. Instead of these operations, particles update themselves with the internal velocity based on particles' self-experiences with their social experiences [19]. It takes real numbers as particles, so it does not require binary encoding as GA. All the trajectories are described with the same number of points (this represents dimensionality of the problem and in our case as shown in Figure 3.10 it is equal to the number of break points shown with red dots). Each member of the population (each feasible trajectory) is informed about the global best result and remembers its own best result.

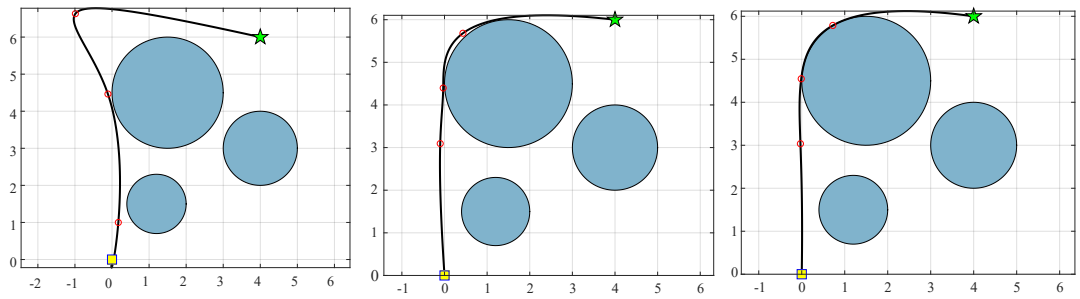


Figure 3.10: PSO in 5^{th}, 25^{th} and 50^{th} iteration [93].

The process of updating continues for a predefined number of iterations or until a termination criterion is met. PSO in trajectory planning aims to find the optimal or near-optimal trajectory by iteratively updating the particle positions and velocities. For the detailed in-

formation about the update, see [19]. Although they demonstrate effective convergence as compared with traditional algorithms, PSO approaches are not reliable in real-time robot applications and they require the number of break points (dimensionality of the problem) to be defined in advance.

Ant Colony Optimization (ACO) [94] is a population-based search technique for the solution of combinatorial optimisation problems, inspired by the fact that ants tend to find the shortest path between their nest and food using pheromone trails instead of vision (they are almost blind). Pheromone trails send signal to other ants. If an ant decides, with some probability, to follow the trail, it will also lay more pheromone and reinforce the trail. The more ants follow the trail, the stronger the pheromone becomes, and it is more likely that other ants will follow it. Pheromone decays over time, and that is the reason why it builds faster on shorter paths. For path finding applications, the free configuration space is filled with randomly placed points representing nodes in a connection graph. Each node is connected to other nodes through edges. The cost (length) and amount of pheromone for each edge are defined and stored in a matrix. When an ant takes an edge connecting two nodes, it leaves an amount of pheromone on that edge. The amount of pheromone left is inversely proportional to the distance (cost) of the edge. This update rule helps consider the vaporization of pheromone between updates. The probability that an edge will be included in an ant's path is calculated based on the amount of pheromone and a heuristic value. This probability is then used in the Roulette Wheel Selection method [95], a technique for choosing edges based on their probabilities. The main shortcoming of ACO is that it needs a long time to reach optimal value if the size of the problem is large.

Although neural networks have some properties that are beneficial for path planning, knowledge representation and extraction are difficult. On the other hand, fuzzy logic is able to handle imprecise and uncertain information but has a drawback because it lacks systematic methodology for their design. Combination of neural networks and fuzzy reasoning refers to hybrid (neuro-fuzzy) systems. In these systems, fuzzy logic can be trained by learning algorithm from neural network theory [96], or fuzzy logic is encoded with neural network [97] or fuzzy logic rules are extracted by neural network [98].

In our specific case, the industrial environment remains constant, and there is no requirement for real-time decision-making or unpredictable obstacle avoidance techniques. Our primary focus is to calculate trajectories within a reasonably short time frame while ensuring that they are executable. Given these considerations, classical sampling-based approaches that search for paths directly in the robot's configuration space are the most suitable planners for our use case. In the following sections, we present an evaluation of multiple variations of sampling-based methods. We assess their performance and propose a methodology for selecting the most suitable planner based on our specific requirements.

3.1.2 Evaluation of motion planners

In recent years, considerable progress was made in making fast and robust algorithms for autonomous robot motion planners [99] with their goal to enable robots to automatically compute their motions from descriptors of tasks and models acquired from sensors [75]. Therefore, many planning algorithms are readily available in different software packages, for example in MoveIt! [100]. Planning libraries, such as the Open Motion planning library (OMPL) [58], provide geometric and control-based planners. Other options exist, for example the Stochastic Trajectory Optimization for Motion Planning (STOMP) [101], Gradient Optimization Techniques for Efficient Motion Planning like Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [102], or Search-based planning library (SBPL) [103].

Since Robotic Operating System (ROS) [104] has gained a lot of popularity in modern robotics and has a potential to become one of the most widely used frameworks, we used only the tools that are compatible with it. This includes MoveIt! [100], RViz for visualization, and Planner arena [59] for benchmarking of different planners. We tested different motion planners available within the Open Motion planning library (OMPL) [58], which offers also constrained planning [105]. The goal of the evaluation was to find planners which can generate collision-free motion trajectories in a fast and reliable manner.

As mentioned previously, the presented example deals with visual quality inspection, where a robot guides the camera to a set of pre-defined postures. The postures are determined so that the camera can see certain aspects of a product/workpiece. Because there are too many postures to reach them in the allocated time, the robot guides the camera to only a small, random sub-set of the given postures. Thus, for every production cycle, we need to generate trajectories between the selected postures. If there are k postures, there are

$$s = k \times (k-1) \tag{3.1}$$

trajectories between them – every posture is connected with every other. One could argue that the connection between postures $(k, k-1)$ is the same as between $(k-1, k)$, and there are actually only half as many. This is spatially true, but not temporally, and the motion needs to be recomputed. This can be avoided if we consider that the sequences can only be increasing, i.e., one can only go from posture 4 to postures 5 and more, and not to postures with a lower number. However, this makes it less general and has other implications, for example, if we actually wanted motion $(k-1, k)$.

In order to determine which planner might be most suitable for the given use-case, we needed to check various parameters. The first criteria was to check if all the planners are able to reach the desired pose. The second criteria was to measure how much time the planner requires to generate the solution. The third criteria we checked was the path smoothness S as introduced in [106]. It is obtained by averaging the squared angles between the consecutive trajectory segments

$$S = \frac{1}{n} \sum_{i=2}^{n-1} \delta_i^2. \tag{3.2}$$

To compute the angles, we again have a sequence of n points $\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_n \in \mathbb{R}^3$ on the path, distributed from the beginning to the end of the path. The angle δ_i between each triple of consecutive points $\{\mathbf{d}_{i-1}, \mathbf{d}_i, \mathbf{d}_{i+1}\}$ was calculated using

$$\delta_i = \arccos\left(\frac{(\mathbf{d}_{i-1} - \mathbf{d}_i)^\top (\mathbf{d}_{i+1} - \mathbf{d}_i)}{\|\mathbf{d}_{i-1} - \mathbf{d}_i\| \cdot \|\mathbf{d}_{i+1} - \mathbf{d}_i\|}\right). \tag{3.3}$$

The forth criteria was the length L of the calculated trajectory. The length of motion given as the sum of n angles traveled by all robot joints between the initial $\mathbf{q}_0$ and final robot configuration $\mathbf{q}_n$

$$L = \sum_{i=1}^{n} \|\mathbf{q}_i - \mathbf{q}_{i-1}\|. \tag{3.4}$$

3.1.3 High level decision making for planning

Machine learning methods have been proposed in different manners for quality prediction. In [107], the authors show that four examined machine learning (ML) algorithms adequately predict quality in injection molding even with very little training data. Pressure

data from the mold cavity was used. Similar was done in [108], where tree-based algorithms, regression based algorithms, and autoencoder were compared. Using a plethora of collected data, the paper shows that autoencoder outperforms other examined tree-based machine learning algorithms in accuracy, precision, recall, and F1-score. Neural networks (NN) were compared to decision trees in [109] for prediction of parts quality in thermoplastics injection molding, showing over 99% accuracy rates. On the other hand, [110] shows that only a combination of NN and support vector machines (SVM) allowed them to reach accuracy above 99%. That work was extended to identify relevant parameters [111]. Since many injection moulding machines are not prepared to access their data on a real-time basis, digitalization of such devices was explored in [112].

In our method, we use the production parameters to train a deep autoencoder (AE). We project the production parameters of a part into a 3-dimensional latent space of a deep AE. There, the known faulty parts are clustered using Gaussian Mixture Model (GMM) with respect to the type of the defect. Each defect type is associated with a camera viewpoint from where its occurrence can be inspected. To determine the most likely defect type of the current part, we check the posterior probability of its latent space projection belonging to the components in the trained GMM. This probability is then used to determine the sequence of the viewpoints where the robot guides the camera to check for the defect, as presented in Algorithm 3.1. Note that only potential defects where the posterior probability is non-zero are checked. Thus, time is saved with less checks and with a higher probability of immediately checking for the most probable defect.

The proposed defect prediction approach is based on the assumption that similar production parameters will produce a similar output. Therefore, we use the production parameters from previously produced parts and use them to predict the outcome of the production process for each new produced part. Another assumption is that there is a subset of most common defects, which cover the large majority of all cases.

Hereinafter, we explain in more detail the database of production parameters, the training of the AE and the clustering on the injection-moulding industrial example from a car-parts manufacturer.

Algorithm 3.1: Procedure for inspecting the viewpoints of one product based on the process parameters

procedure Check viewpoints
  project new part parameters (Tab. 3.1) into the latent space θ^{AE}
    of a previously trained AE;
  compute posterior probabilities $\mathbf{P}$ of the established c
    GMM clusters associated to c most common defects;
  arrange $\mathbf{P}$ in descending order;
  set $i = 1$
  for probabilities $p \in \mathbf{P}$:
    choose viewpoint(s) $\mathbf{V}_i$ corresponding to p_i;
    plan trajectory between current robot posture
      and $\mathbf{V}_i$;
    execute trajectory and set $i = i + 1$;
    if vision check (Fig. 3.22) returns error:
      discard product;
      break
  end

Database

We had at our disposal a database that consisted of production parameter data for 4436 good (OK) parts and 45 parameter data for faulty (NOK) parts. We also had 32 faulty physical parts with associated parameters available. We refer to this database as DB1.

The main reason for many more entries for good parts is that a typical industrial production process produces significantly more good than bad parts. In addition, in many production lines including the one studied in this paper, intermediate quality checks are performed. Since the parts that reach the end of the production line have already passed these checks, the number of bad parts is reduced. Those that are deemed faulty at the intermediate checks are discarded and their production data might not be stored or is incomplete. In any case, the data about faulty parts identified at the end of the production line can initially only be collected by assigning a human worker to manually inspect the parts. Thus in general, data for faulty parts is difficult to obtain.

To obtain the production parameters for each part, their QR code identifier is scanned and the data extracted from the production line database. In our practical example, the discrepancy between the number of faulty physical parts and the number of data for faulty parts in database DB1 is due to some parts having the (small) QR code identifier not readable.

During the production process, hundreds of production parameters are stored for every product. However, a large portion are not relevant for the process itself (server name, user name, folder, etc.). In collaboration with a production engineer from the manufacturer, we identified 22 relevant production parameters. They are listed in Table 3.1. Some are easily understood, while the others, such as PD_DIA average and minimal values, were proposed by the engineer.

Table 3.1: Injection Moulding Parameters Used for Prediction of Defects.

cycle time	switch after volume
injection time	machine cycle counter
max injection pressure	feeder housing temperature
switch after pressure	tool 1 heating circuit
dozing time	tool 2 heating circuit
material cushion	tool 4 heating circuit
tool 7 heating circuit	blasting nest
tool 10 heating circuit	nest insertion
tool 11 heating circuit	nest withdraw
cooling nest	PD_DIA_min. value
marking nest	PD_DIA_avg. value

Defect Probability Prediction Using a Deep Autoencoder

As presented in Algorithm 3.1, we pass the data of DB1 through a deep autoencoder. The data was normalized for each parameter. The autoencoder was designed empirically, with a 22-dimensional input and output layers and 5 hidden layers of the size 15, 10, 3, 10, 15, respectively. The hyperbolic tangent sigmoid activation function was used for the neurons. The autoencoder passes its input data to the output, so that the output data matches the input data with the highest possible precision. The data is pushed through differently sized layers, including through the lowest-dimensional layer, known as the latent space. The data in the latent space now encodes the data with the smallest amount of parameters. From

the manufacturer we know that there are 5 typical errors that can occur. These include porous material, faulty/broken sides, incomplete bottom edge, fault/broken pins and faulty windings as a result of the injection moulding process. The defects are depicted in Figure 3.11. We therefore partitioned the data into $c = 5$ clusters. In order to exclude the effect of the clustering method on the clustering result, we tested two clustering methods. These were k-means and Gaussian Mixture Model (GMM) clustering, with 5 target clusters for each. The former, which is the most commonly used clustering method, forms spheres around the centers of the clusters and performs hard classification, while the latter can handle non-round shapes of clusters and perform soft classification [113].

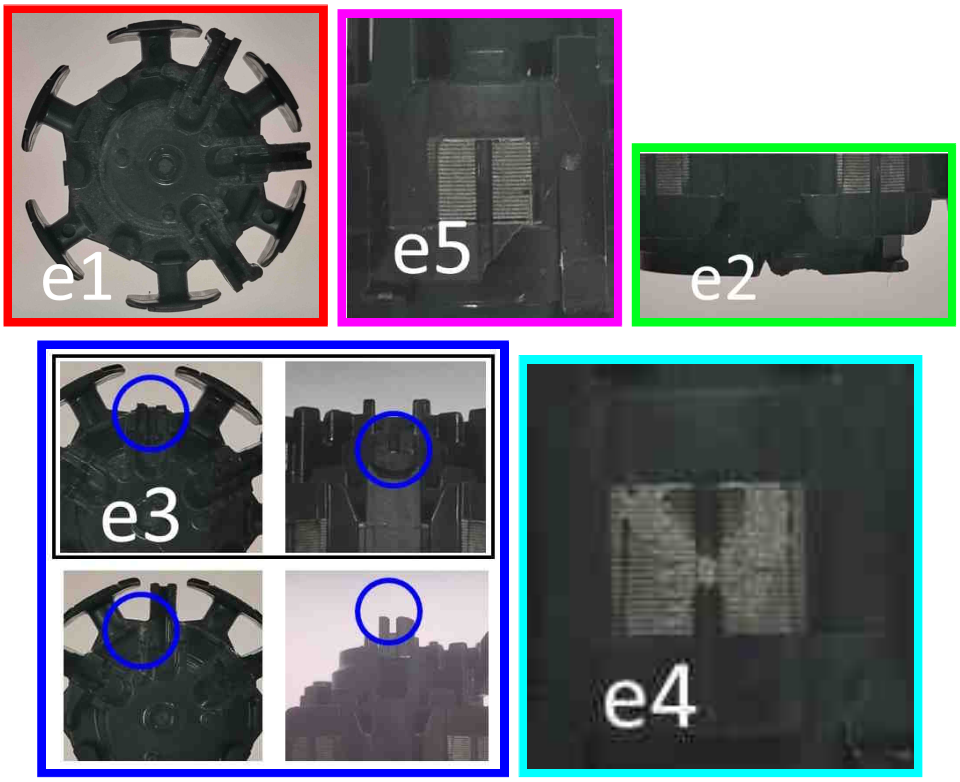


Figure 3.11: Five typical defects: e1) porous material, e5) faulty/broken sides, e2) incomplete bottom edge, e3) faulty/broken pins and e4) faulty windings as a result of the injection moulding process.

3.2 Experimental Evaluation and Results

In this section, we first present our experimental setup. We proceed with the results of benchmarking the motion planners. In order to know from which point the object needs to be inspected, we need a defect prediction method. Thus, we finally show the results of defect prediction introduced in Sec. 3.1.3.

3.2.1 Experimental setup

The experimental setup consists of the work-cell where finished products arrive for inspection after injection moulding. UR 10 robot is holding industrial camera and moves it to the inspection points from which the object is inspected for the defects. Experimental setup is shown on the left side of Figure 3.12 and corresponding RViz simulation is on the right side.

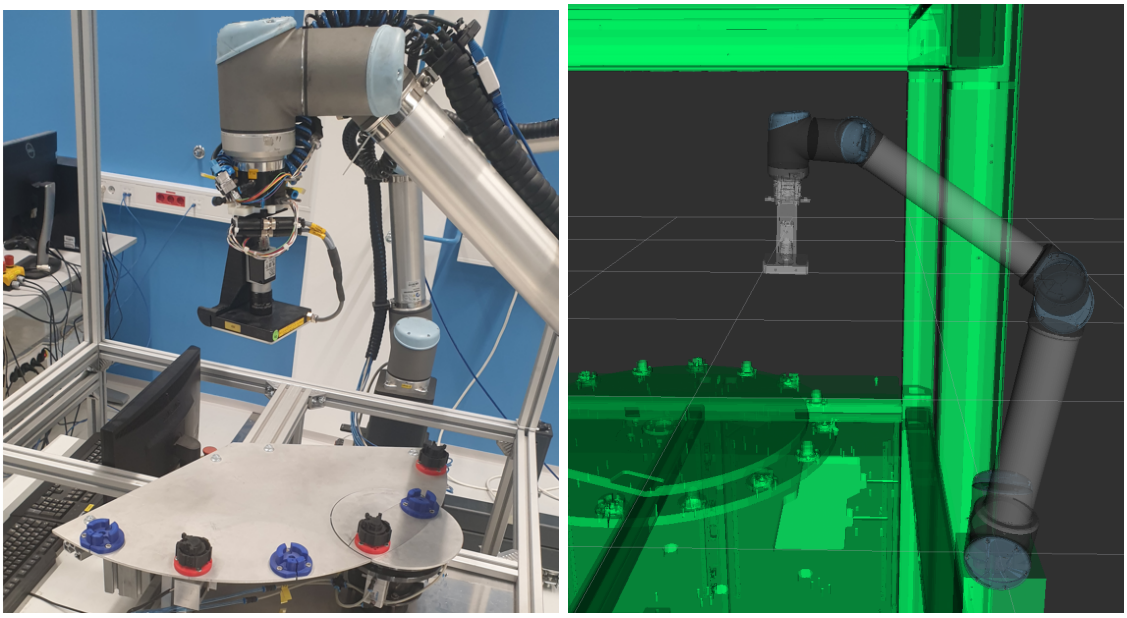


Figure 3.12: Physical implementation of the visual quality inspection cell for the use-case (left), and RViz visualization (right).

3.2.2 Motion planner evaluation

In our motion planner method evaluation, we evaluated all of the 21 different motion planners available within the Open Motion Planning Library (OMPL) [58]. The planners are listed in Table 3.2. We have considered the example with 11 postures. Some of the postures are presented in Figure 3.13. Because the product is relatively small, some postures are quite close together and those are not presented. The potential defects were

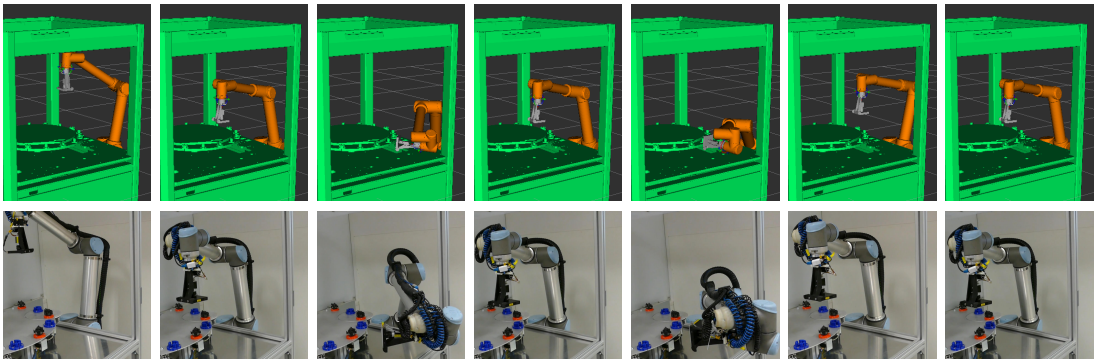


Figure 3.13: Seven out of eleven postures for visual quality inspection from the given use-case in RViz visualization and in real-world execution.

associated with predefined viewpoints which were set so that they ensure the camera-object relation such that the images are sharp. Note that some viewpoints have to provide a look at the product from the side and some from the top. In our implementation, a look from the side of the object included two postures, one for the sides and windings and one for the bottom of the product. An external actuated axis was installed into the fixture of the product to ensure that the object can rotate. Thus only one robot posture was required. The rotations of the object were not considered in the evaluation of motion planning, only

the robot motion was considered. Having 11 postures means that, according to Eq. (3.1), one would have to pre-compute and store 110 different motions. This is manageable, but again reduces the generality of the approach and quickly becomes unsustainable with a larger number of postures. In our experiments, we measured four main criteria for the evaluation (ability to reach poses, time required for planning, trajectory smoothness and length of the trajectory). Robot needed to go through all 11 given poses for all the planners. In order to achieve the completeness of the results, experiments were repeated 250 times.

Table 3.2: List of tested OMPL planning algorithms.

Pl. #	Planner name
1	BiEST (Bidirectional version of EST)
2	ProjEST (Projection-based version of EST)
3	LazyPRM
4	LazyPRM*
5	SPARS (SPArse Roadmap Spanner algorithm)
6	SPARS2
7	SBL (Single-query, Bi-directional and Lazy Collision Checking)
8	EST (Expansive Space Trees)
9	LBKPIECE (Lazy BKPIECE)
10	BKPIECE (Bidirectionnal KPIECE)
11	KPIECE (Kinematic Planning by Interior-Exterior Cell Exploration)
12	RRT (Rapidly-exploring Random Trees)
13	RRT Connect
14	RRT*
15	T-RRT (Transition-based RRT)
16	PRM (Probabilistic roadmap)
17	PRM*
18	FMT (Fast Marching Tree algorithm)
19	PDST (Path-Directed Subdivision Trees)
20	STRIDE (Search Tree with Resolution Independent Density Estimation)
21	BiTRRT (Bi-directional Transition-based RRT)

First, we checked if the planners reach the desired pose. This is shown in Table 3.3, where it can be seen that some planners did not converge to the solution in all 250 iterations. These were immediately ruled out.

Table 3.3: Success rate of planners for 250 planning attempts for 110 transitions between two postures.

#	1	**2**	3	4	5	6	7	8	9	10	11
%	100	100	100	100	98	66	100	100	100	100	100
#	12	13	14	15	16	17	18	19	20	**21**	
%	99	100	99	98	100	100	100	100	100	**100**	

Subsequently,we shifted our focus to a crucial aspect in optimizing production: cycle times. Since short cycle times are needed to prevent bottlenecks in the production, the

average time required for planning a new trajectory was checked. As illustrated in Figure 3.14, some of the planners required significantly longer durations (times greater than 2 seconds are not shown for data clarity), and these were eliminated as the objective was to generate trajectories as quickly as possible. It is important to mention that all our calculations were performed on a computer equipped with an AMD Ryzen Threadripper PRO 5975WX processor and 512GB of RAM. The planners presented in the figure are numbered and correspond to the names listed in Table 3.2.

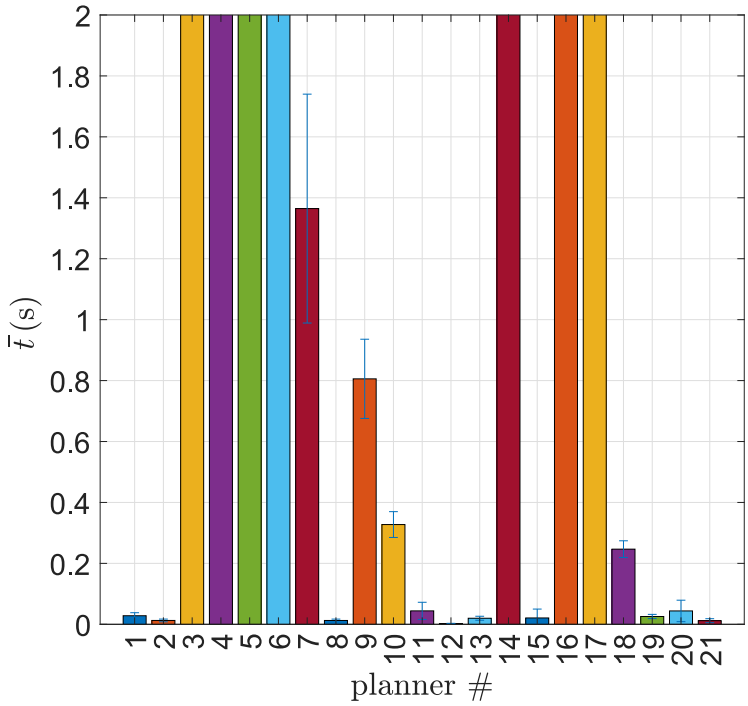


Figure 3.14: Average duration for the calculation of transition trajectory between 11 random pairs of poses, 250 times per pair, for 21 motion planning algorithms listed in Table 3.2. The times are cut at 2 seconds, while the calculations for some planners took longer, up to pre-defined 5 s calculation cut-off time.

We also evaluated the smoothness of the trajectories. This benchmark criteria was computed using Equations (3.2)-(3.3). The results are shown in Figure 3.15. It should be noted that these values refer to the smoothness of the plan itself and consider only positions between the points of the trajectory when measured at the end of the robot. Since some planners try to follow a straight line for as long as possible, some of them always report zero smoothness, or they have only two points in the initial solution and, therefore, smoothness cannot be calculated and is assigned value zero.

Finally, we analyzed the length of the generated trajectories, as expressed by Eq. (3.4), and this information is presented in Figure 3.16. It is worth mentioning that there is the option to simplify the path, which we tested across all the planners. We tested it for all the planners, and after simplification, all of them reported similar length results at 3.06 rad and standard deviation of 2.09.

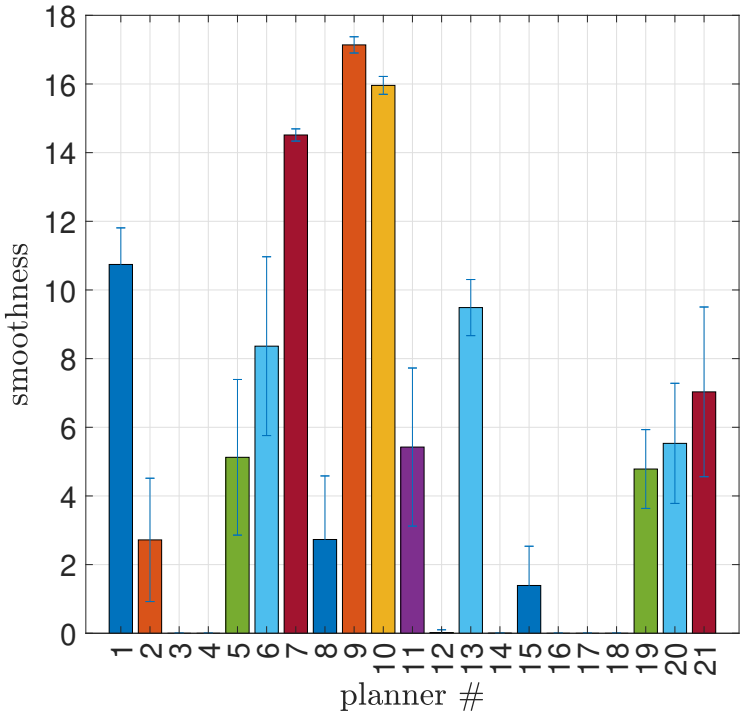


Figure 3.15: Average smoothness (Equations (3.2)-(3.3)) of the generated trajectories between 11 random pairs of poses, 250 times per pair, for 21 motion planning algorithms listed in Table 3.2.

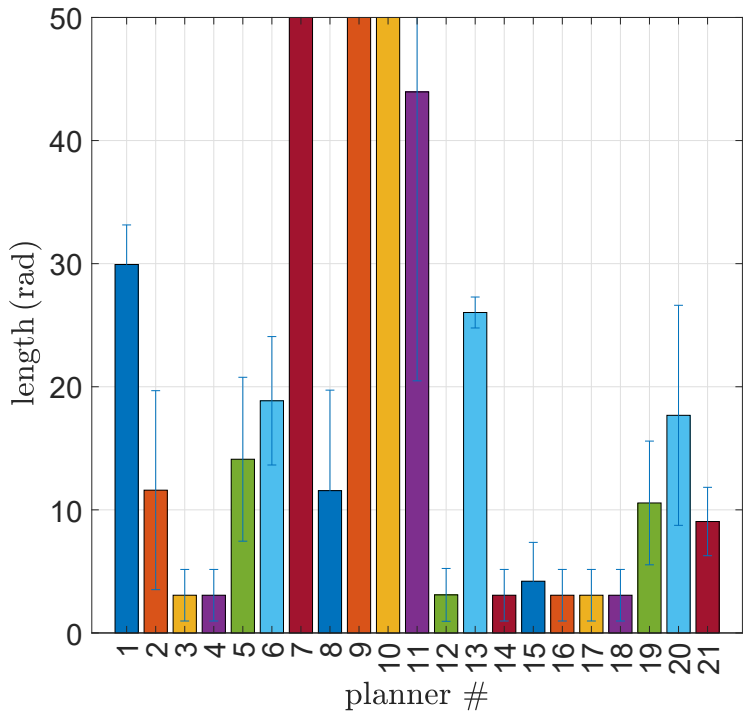


Figure 3.16: Average length (Eq. (3.4)) of transition trajectory between 11 random pairs of poses, 250 times per pair, for 21 motion planning algorithms listed in Table 3.2. The lengths are cut at 50 rad.

Considering the results in Table 3.3, and Figure 3.14, we can see that planner #21, Bi-directional Transition-based Rapidly-exploring Random Trees (BiRRT), which combines the exploratory strength of RRTs with the efficiency of stochastic-optimization methods (e.g., Monte Carlo optimization), and also improves performance compared to the Extend T-RRT in terms of success rate and running time [114] performed best in the given use-case. The chosen Bi-TRRT converged to a solution all the times, was very fast and the resulting trajectories were smooth and short. Variations of RRT, such as T-RRT performed practically on the same level if not better, but did not converge in completely all of the planning attempts. Planners #1, 2, 19 and 20 were excluded because of longer and less smooth trajectories, even though the results were very close to the chosen planner.

3.2.3 Decision making

In order to obtain the goal positions towards which the robot needs to move, we created a decision making method that predicts what kind of a defect occurred on the product. For this purpose, AE network described in Section 3.1.3 was trained on the data given in DB1. Database projected to the latent space for all the parts is shown in Figure 3.17. Latent space projections of the parameters for the good parts are marked with black and for the faulty parts with red.

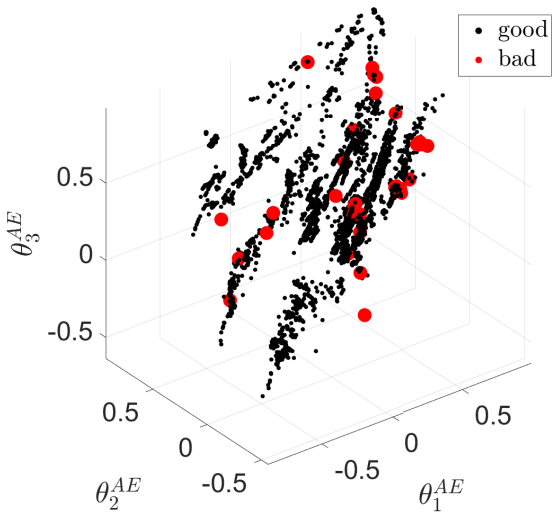


Figure 3.17: Database DB1 datapoints projected into the 3-dimensional latent space of a deep AE with red dots marking datapoints for 32 physically available faulty parts and smaller black dots for good parts.

The results of clustering using both (k-means and GMM) methods are shown in Figure 3.18. Clusters are marked with different colors. Except for one part, both algorithms clustered the data in the same manner.

To check if the clustering depends on the autoencoder, we trained the deep AE with different random initial parameters of the deep AE network. Clustering in the latent space of 6 autoencoders (each trained with different initial parameters) is shown in Figure 3.19. The clustering results are the same in all cases, showing that training of deep AE with different initial parameters will change the latent space, but not the clustering.

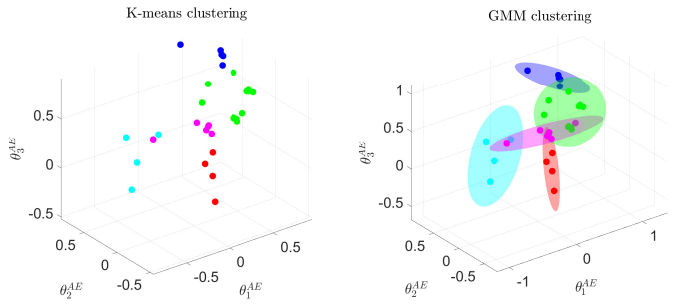


Figure 3.18: Clustering of data points in the 3-dimensional latent space of AE using K-means (left) and GMM (right) clustering algorithms.

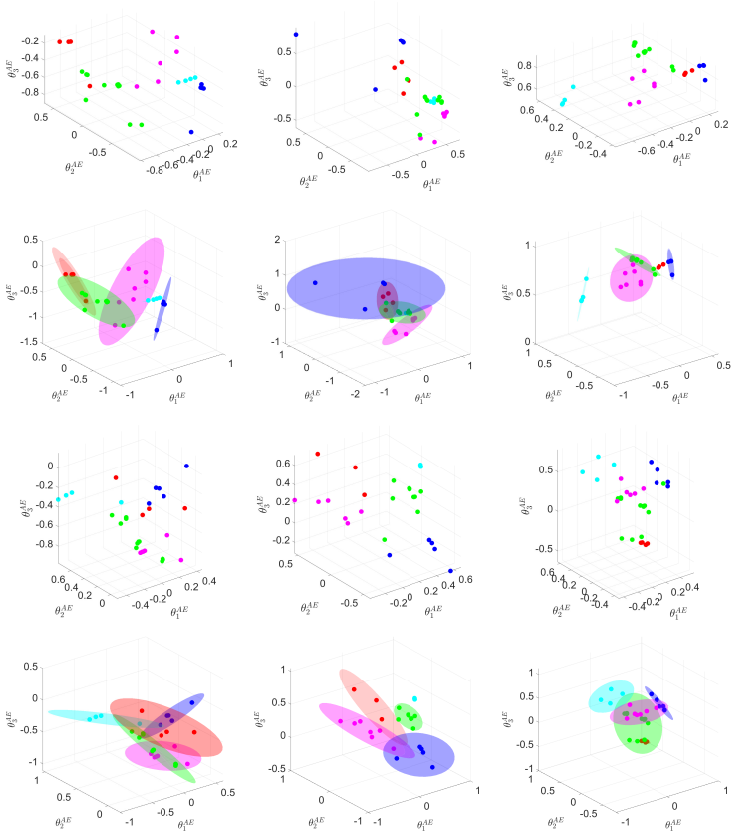


Figure 3.19: Clustering of NOK datapoints in latent space of AEs trained with different initial seeds. In all cases the clustering is the same as in Figure 3.18.

By associating the latent space datapoints with the physical parts with known defects, we can see in Figure 3.20 that these indeed match the described defects, except for one that fits into two clusters (green and purple, shown in Figure 3.20 in the 5-th row left). This result confirms that the clustering procedure is appropriate. Thus, for any new datapoint, we check to which cluster it belongs using the posterior probability for all GMM components in the latent space projection. Results for posterior probability of all DB1 datapoints for all components are shown in Figure 3.21. Our approach proceeds with checking the parts in the descending order of probabilities of possible defects. This means that the motion of the robot is potentially different for each part that needs to be inspected. We therefore generate the robot motions that place the camera at the required viewpoints online using a suitable motion planner.

Figure 3.20: Manual clustering of physical parts per defect coincides with the clustering in the latent space of the AE, depicted by the color edges and colors of clusters in Figure 3.18.

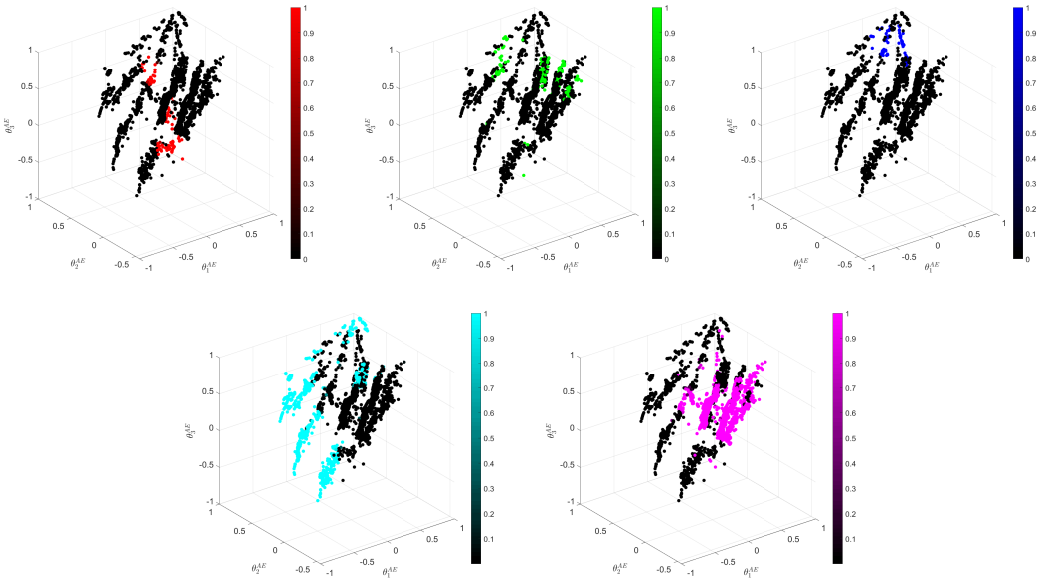


Figure 3.21: Probability of defect determined by posterior distribution of GMM components. The colors of the highest defect probability are associated with defects in Figure 3.11.

3.3 Discussion

In this chapter, demonstrating an example of the task adaptation strategy, we have shown a method of accelerating the industrial process using the combination of NN-driven high-level decisions and motion planners for generating the trajectories that enable the successful execution of the task.

The presented defect prediction approach mostly depends on the available data. With thousands of datapoints for good and faulty parts, it is possible to increase the trust in the prediction of possible defects. Companies, however, tend to produce orders of magnitude more of good parts than faulty parts. Thus, collecting a large database, especially at the end of the production process where the parts that reached this point have passed the intermediate quality check is not easy. As can be seen in the practical production example considered in this paper, the database of good and faulty parts (DB1) can be extremely lopsided.

Our approach checks all the parts after the final production stage, but uses the prediction model to change the order of inspections and only checks the most likely defects.

Given that in our approach, the robot must frequently move between different pairs of viewpoints, we tested which of the available motion planners is best for our practical production process. There is no motion planner that would be best in every situation, as also stated in [59]. For the considered production process, extensive statistical evaluation has shown that several planners might be appropriate. The best results were obtained with Bi-directional Transition-based Rapidly-exploring Random Trees (Bi-TRRT) method. For the considered production process, learning 110 trajectories and then recalling them from the database is still viable. Nonetheless, it is easy to imagine visual inspection processes where many more motion trajectories would need to be generated. This is, however, not necessary because the available motion planners can generate trajectories for visual quality inspection in a fast and reliable manner.

For completeness of the adaptive visual quality inspection approach, we implemented a deep neural network classifier that takes the image as the input and classifies the part as good or faulty at the output. The results of this classification for the incomplete bottom edge defect are shown in Figure 3.22.

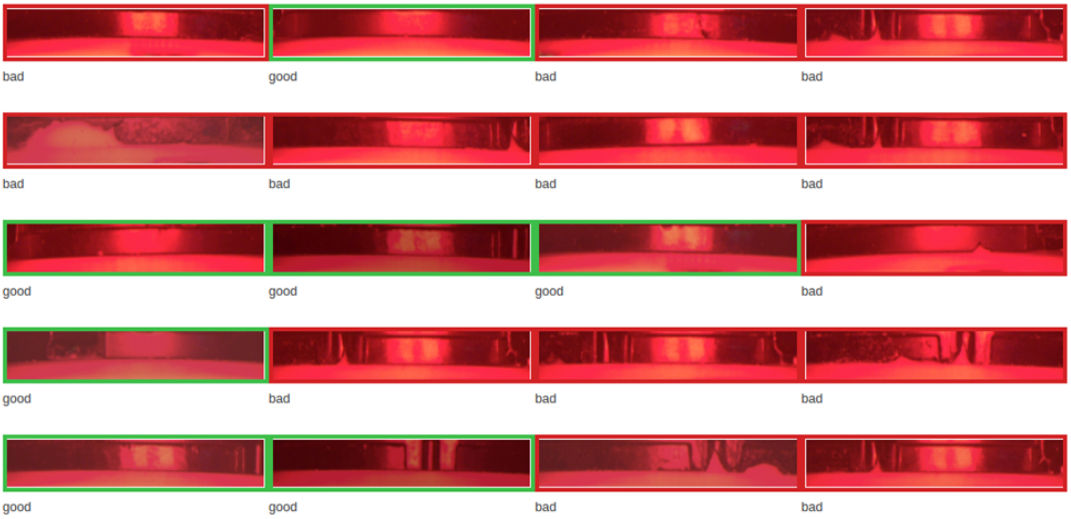


Figure 3.22: Results of final visual quality inspection for the "incomplete bottom edge" (e2) defect of several parts.

In this chapter, we presented an approach that takes the production parameters of a real-world injection moulding production line to predict potential defects of the produced parts. We have shown how to use autoencoders and clustering of datapoints to predict what kind of defect might be present. The evaluation has shown that the clustering and classification do not depend on the training of the AE networks. We have also analyzed different motion planners to generate motions that bring visual inspection cameras to the required viewpoints in a collision-free manner.

With this method, we closed the feedback loop between the prediction of the defect and the camera viewpoint. For our future work, we plan to create a method that will be able to autonomously change the production parameters based on the observed defects. However, considerably larger databases of both good and faulty parts and also types of defects will be required to achieve this.

It should be noted that this approach is in the middle of the pyramid and that the trajectories generated with this method are not optimal. This means that further optimization of the generated and stored trajectories can be done (for instance, using the method from the previous chapter), allowing us to make the inspection process faster. Moreover, this is still only adaptation of an industrial process in a known and structured environment where the robot does not interact physically with an object, thus allowing us to make an accurate digital twin of the system in simulation by considering only a geometry of the scene.

Chapter 4

Dynamic Task and Process Behaviour Generation

In the preceding chapter, we have shown an example of the process adaptation when generating the trajectories. In that case, a robot did not physically interact with the environment and we had a lot of knowledge about the task - position of obstacles, final end-effector positions, potential errors and a fixed number of required trajectories. The bottom of our pyramid (Figure 4.1) shows the most complicated case of the robotic learning because the objective of dynamic task generation is to acquire a comprehensive skill set for an infinite range of targets by interacting with the surroundings without relying on an explicit environmental model.

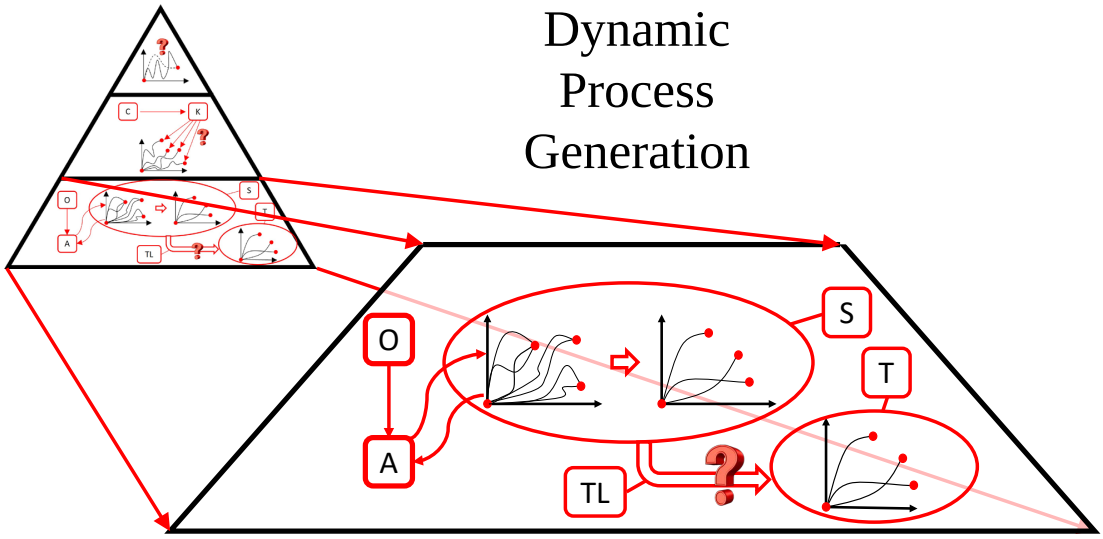


Figure 4.1: Trajectory generation is set to be in the bottom of our classification pyramid since it is rarely applied in commonly used robotics due to its complexity.

In this case, we only know which action the robot needs to perform - labelled with O in Figure 4.1. We have a rough and highly inaccurate model of the environment in the so-called source environment (this is usually simulation). By performing RL, we search for the appropriate trajectory parameters (labelled with A in Figure 4.1) corresponding to each

instance of the action O in the source environment (labelled with S). Once we have sufficient knowledge about the robot behaviour in the roughly modelled source environment, we begin a process of transferring it to the more accurate target environment (T) which is usually the real-world with a real robot. This process is called transfer learning (TL) using domain adaptation.

Many review papers [23], [24], [115], [116] have shown the possibility of RL algorithms to successfully obtain task knowledge from scratch. Since the algorithms presented in those review papers usually require a huge amount of data and a lot of iterations to converge successfully, their usage is limited to simulation environments. The perfect simulation does not exist, and therefore the knowledge obtained this way cannot be directly applied to the real-world, but has to be adapted. This is why, in this part of the thesis, we put the main focus on the methods for adaptation of this knowledge - Transfer Learning (TL).

We demonstrate our approach on the robot throwing action. We have chosen this task since it was previously already considered in RL settings [117] and because it can be relatively accurately mathematically modelled. It should also be noted that this is an example of the dynamic task and that the robot was throwing using the throwing spoon in its hand. This way, the ball is not firmly attached and we cannot control the release point, but the ball detaches itself when the arm acceleration becomes negative. We present two similar approaches for performing TL with two different source environments and two different robots. Our approaches are based on updating NN where the source environment knowledge is stored and the main focus is put into accelerating this procedure using RL algorithms. Since there is a relatively large amount of testing performed, and for the sake of the result repeatability, we use (different) simulations as both the source and target environments.

In addition to this, as a separate subsection, we provide benchmark of two additional methods for storing the source environment knowledge using probabilistic approaches. Although we do not perform TL, it provides a great baseline for the future work that would require less user knowledge input into the domain adaptation itself.

4.1 Methodology

Our method relies on utilizing the capacity of neural networks (NN) to retain knowledge from the source environment. Further, the network is updated using a combination of reward-weighted policy learning algorithm (RWPL) that we introduced in Sec. 2.1.3 and backpropagation with data obtained from the target environment.

We demonstrate two approaches, but in both cases, we use NN to store the acquired knowledge from the source environment. In the first case, we extend our method with using encoder-decoder type of NN that extracts only important features from the learnt behaviour. The knowledge itself is not stored in AE network, but it enables us to use another smaller network for storing the knowledge. In the second case, we train only one network that stores the whole end-to-end knowledge and we update only one layer.

4.1.1 Trajectory acquisition and encoding

To demonstrate our transfer learning (TL) approach, it was necessary to acquire proficiency in the source environment. Specifically, for the throwing action, we needed a suitable dataset denoted as:

$$\mathcal{D}_{\text{so}} = \{\mathbf{W}, \boldsymbol{T}\}, \tag{4.1}$$

where $\mathbf{W}$ represents the trajectory parameters, while $\boldsymbol{T}$ corresponds to the landing spots. Since all our throws were executed in the sagittal plane in front of the robot, the landing

spots are always two-dimensional $\boldsymbol{T} \in R^2$. The vector $\mathbf{W}$ signifies a parameterized trajectory, and its dimension depends on the chosen encoding parameters. Encoding plays a crucial role as it ensures that all trajectories share the same size, regardless of their length. This uniformity is essential for training the neural network, which requires a fixed number of inputs and outputs.

As mentioned before, our methods have different source environments and they are performed on different robots: Mitsubishi PA10 and PAL robotics TALOS. The first method includes also a separate AE network for dimensionality reduction, while the second method relies on changing only one layer of the network.

Throwing with robot arm

For the first method, 7 DoF Mitsubishi PA-10 robot equipped with a throwing spoon was used. The knowledge in the source environment was not acquired by performing RL, but given mathematically. Assuming that the robot is directed towards the target (that its orientation is correct), the throwing problem can be considered as a planar problem in the sagittal plane of the robot. Because of this, targets could be described by only two parameters describing the distance from the base in x and z axis ($\boldsymbol{T} = [T_x, T_y]$) and only three joints that contribute to performing a throw were kept active and taken into account when calculating inverse kinematics during the trajectory execution. The method of calculating the database is presented in detail in [61]. Although the dataset was initially represented in joint space, we aimed for our approach to be general and applicable to various types of robots. Therefore, we transformed all the calculated trajectories into Cartesian space using forward kinematics. Lastly, we utilized CDMP parameters for encoding the trajectory:

$$\mathbf{W} = [\boldsymbol{w}_i^p, \mathbf{g}^p, \mathbf{y}_0^p, \boldsymbol{w}^o, \mathbf{g}^o, \mathbf{y}_0^o, \tau]. \tag{4.2}$$

Here $\boldsymbol{w}_i^p \in \mathbb{R}^3,\ i = 1, \ldots, N$, defines the shape of the position part of the trajectory, $\mathbf{g}^p \in \mathbb{R}^3$ defines the final position of the trajectory and $\mathbf{y}_0^p \in \mathbb{R}^3$ defines the initial position of the trajectory. Similarly, $\boldsymbol{w}^o$ describe the orientation aspect of the trajectory. Final and starting orientations are presented with quaternions, and therefore they are in a four-dimensional space: $\mathbf{g}^o \in \mathbb{R}^4$ and $\mathbf{y}_0^o \in \mathbb{R}^4$.

Throwing with humanoid robot

In the second method, we employed the left arm of PAL Robotics' humanoid robot TALOS, which was equipped with a throwing spoon. Unlike the first method, all 7 DoF of the arm were utilized to execute the throwing action. In this case, the dataset was too complex to be pre-calculated, so the knowledge about the task had to be autonomously acquired during the learning process. In order to accelerate learning in the source environment, we gave some knowledge about the task - one throwing trajectory defined as a part of parabola to be executed. We encoded this parabola into CDMP and got parameters as in Eq. (4.2). Having the initial trajectory encoded in such parametric representation, we were able to start modifying it by adding random noise on parameters describing position, $\boldsymbol{w}_i^p$, orientation $\boldsymbol{w}_i^o$ and the time of trajectory execution τ. To ensure that the trajectories remain within the same plane and maintain consistent throw direction, we specifically modified the weights that determine the position along the x and z axes and the rotation around the y-axis. However, during this process, we encountered challenges due to the narrow workspace of the TALOS robot when performing actions above its head. Even minor adjustments to the trajectory resulted in singularities or joint limit issues, limiting our ability to modify these specific parameters effectively.

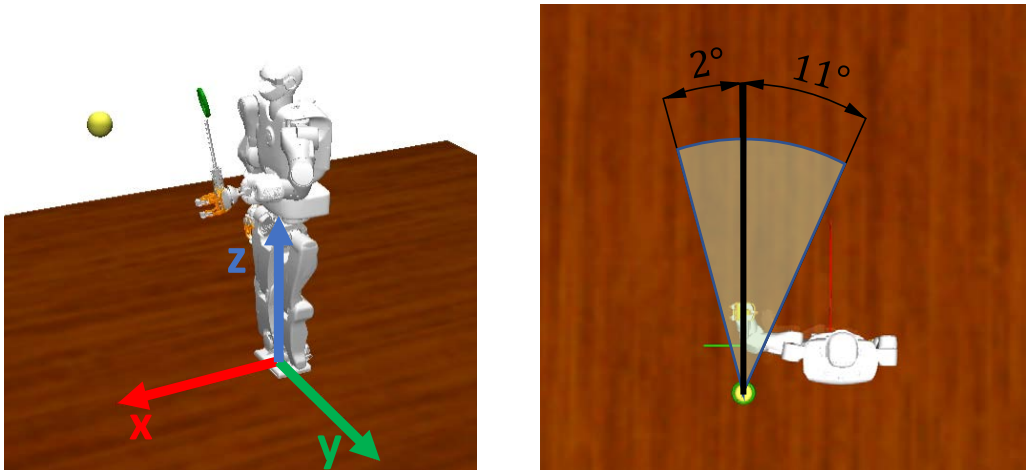


Figure 4.2: Left image shows the initial throwing action and coordinate system of the robot and right image shows the feasible range of angles with the origin of the arc put into the initial position of the throw.

In order to overcome these problems, we need to position the trajectory differently [62]. This is why we decided to try rotating it around its initial point (Figure 4.2) and finding at which angle the number of feasible trajectories is the highest. The initial position could be rotated for 2° counter-clockwise and for 11° clockwise without the robot reaching any joint limits, so this was taken as the range in which the optimal orientation needs to be found. In order to check which trajectories are executable and which not, our first step was to rotate the initial trajectory. We generated 53 trajectories in the range $[2°, -11°]$ with rotating the initial trajectory for up to 2° counter-clockwise and up to 11° clockwise with the step of 0.25°. In further text and images, we will use the positive sign for counter-clockwise direction and negative for the clockwise. This set of trajectories was then encoded in CDMP. When having the initial trajectory rotated in different angles with respect to initial position and encoded in CDMPs, we were able to start executing and measuring different variations of the task. In order to do that, we needed to modify forcing terms for position and orientation for all 53 trajectories. Parameters that describe the forcing term are weights $\boldsymbol{w}_i^p$, $\boldsymbol{w}_i^o$. Since we want trajectories to stay in the consistent direction, we again modify only the weights defining the positions in x and z axes and rotation about y-axis. For all the c reachable points of these trajectories, we calculated manipulability measure and distance to joint limits. Manipulability is a measure of manipulating ability of robotic mechanisms in positioning and orienting the end-effectors [118]. This number describes the robot's ability to change its end-effector position given the current joint configuration. The greater it gets, the greater range of feasible motions can be performed from that configuration. If this number is zero, it means that the robot is in a singular configuration and it cannot generate velocity in one of the Cartesian directions. This measure is defined for each c-th point in the dataset as:

$$\mu_c = \sqrt{det(\mathbf{J}\mathbf{J}^T)}. \tag{4.3}$$

In our case of a redundant robotic arm with 7-DoF, the Jacobian matrix is of size $6x7$. Since throwing is performed only in one plane, we are not interested in the ability to move in the direction of y-axis and in rotations about x and z axes. This is why we used reduced Jacobian of size $3x7$ where we left out corresponding rows of the matrix. We calculated normalized values of manipulability index ($\hat{\mu}_c = \mu_c / \max(\mu_c)_{c=1}^{C}$) for all the

executed points in the database. Distance from the joint limits is the second but equally important parameter for successfully executing the trajectory. As a measure of distance from the joint limits, we used the smallest normalized distance to the joint limit from all the 7 joints of the robot arm. For all C points in the dataset it was calculated as:

$$\begin{aligned} \rho_c &= min(j_i)_{i=1}^{7}, \\ j_i &= \frac{min(l_{i,1}, l_{i,2})}{|\theta_i^{max} - \theta_i^{min}|}, \\ l_{i,1} &= |\theta_i^{max} - \theta_i|, \\ l_{i,2} &= |\theta_i - \theta_i^{min}|, \end{aligned} \tag{4.4}$$

where θ_i^{min}, θ_i^{max} and θ_i are lower, upper limit and the current state of the i-th joint, respectively. Once we calculated all the manipulability measures μ_c and joint limit distance ϕ_c, we also calculated the cross section of these two values:

$$c_c = \frac{\hat{\mu}_c + \rho_c}{2}, \tag{4.5}$$

and this number was used as criteria of how apposite it is for a robot to be in that state and the values are shown in Figure 4.3.

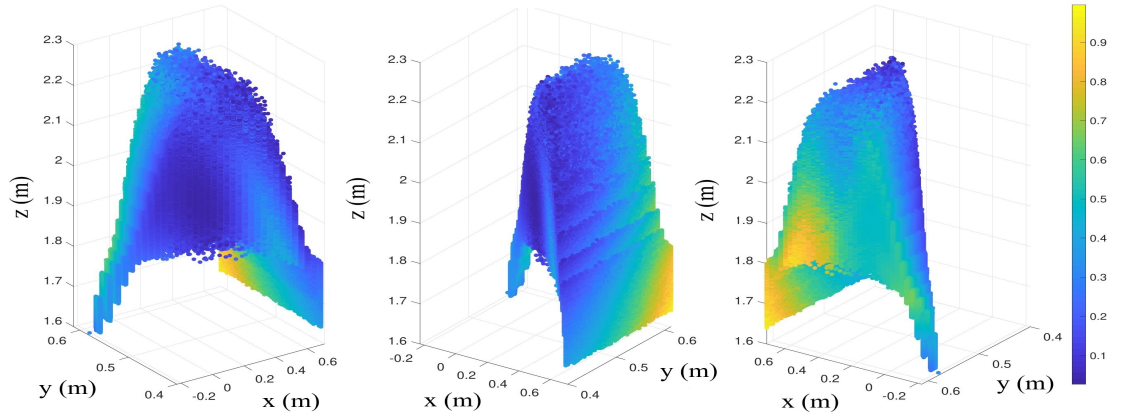


Figure 4.3: Cross section of the normalized manipulability measure and minimal distance to the joint limit for the reachable points.

Since the problem is planar and we can change only the angle of the throw, our next step was to project the mean value of cross section for all the points in the dataset that have unique x and y coordinates to xy-plane.

With having the database of points, we needed to approximate which angle is the best for the throw. In [62], we presented in detail three different approaches for solving this problem. We applied evolutionary algorithm (CMA-ES [119]), used generalization, brute-force search and got to a conclusion that the initial parabola needs to be rotated for $\sim -3°$.

Once we had a good initial trajectory positioning, we encoded it again in CDMP. We were able to start modifying it again by adding random noise on parameters describing position, $\boldsymbol{w}_i^p$, orientation $\boldsymbol{w}_i^o$ and the time of trajectory execution τ. Again, we modify only the weights defining the position in x and z axes and rotation about y-axis. This resulted in having a huge database of trajectories. This time, we were also spawning the ball in the throwing spoon before executing. Trajectories where the ball fell out of the spoon or

fell from the spoon and hit the robot were discarded. As the landing spots we have chosen random (y, z) pairs in the vertical plane of the trajectory. This resulted in again having the source environment database as in Eq. (4.1).

4.1.2 Storing the knowledge in neural network

Having the generated database, given with Eq. (4.1), we can train a NN using backpropagation. The NN thus stores the knowledge from the source domain. In the following subsections, we present two approaches that we developed for our two use cases.

Throwing with robot arm

In the first method, we employed a combination of two neural networks to effectively store knowledge, similar as we did in [120]. As depicted in Figure 4.4, this architecture encompasses an AE network and a more compact network. The underlying concept behind this approach is to condense the dimensionality of our trajectory dataset to a level beyond the capability of CDMPs. Consequently, we store this condensed knowledge within the smaller network, which offers the advantage of faster updates during TL.

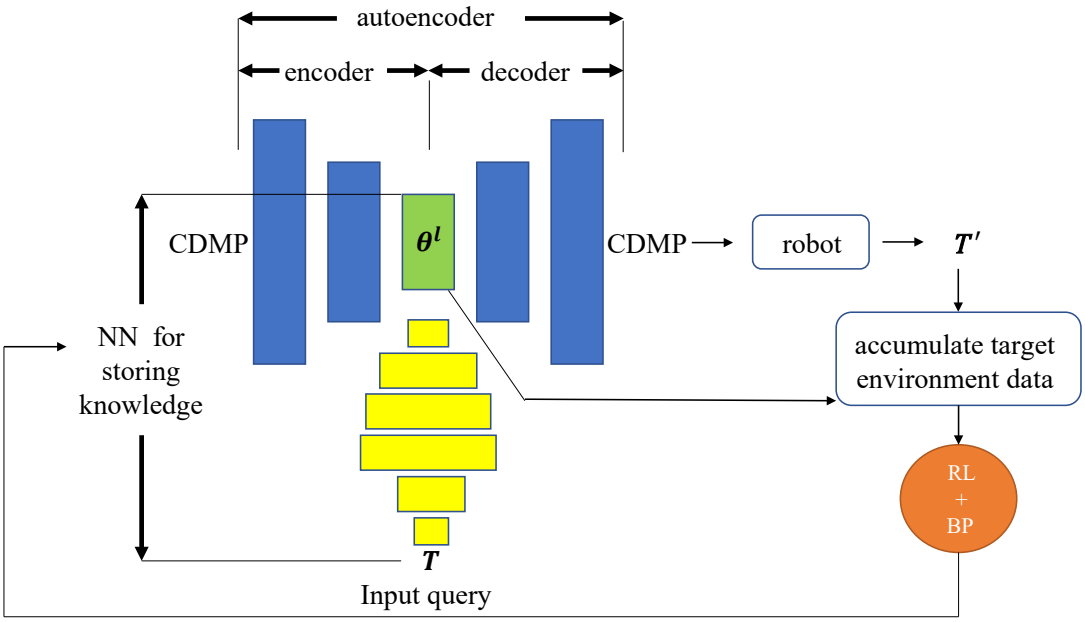


Figure 4.4: Schematic structure of the implemented approach.

Deep AE neural networks are often used for dimensionality reduction. When training the AE network, it learns to copy its input data to the output with the highest possible precision, but data is pushed through different sized layers. The AEs consist of two parts: an encoder and a decoder part. The encoder part is used to push the input data through the network until the bottleneck is reached. This bottleneck is called latent space and it extracts the most important features of the data. The second (decoder) part is used to decode the data from the latent space to original representation with minimal loss of information. In our case, we used 3-dimensional latent space:

$$\boldsymbol{V}^l = [V_1^l, V_2^l, V_3^l]^T. \tag{4.6}$$

As input/output of the network, we used CDMP parameters (Eq. (4.2)) describing the trajectories in Cartesian space in x and z axes as well as the orientation around the y axis together with the parameter describing the time in which the trajectory is executed - τ. This sets the input of AE network to:

$$\boldsymbol{V} = \left[\boldsymbol{\Theta}_x^T, \boldsymbol{\Theta}_z^T, \boldsymbol{\Theta}_{rot_y}^T, \tau\right]^T, \tag{4.7}$$

$$\boldsymbol{\Theta}_{\{.\}} = [\boldsymbol{w}_{i,\{.\}}, y_{0,\{.\}}, g_{\{.\}}], \tag{4.8}$$

with y_0 the starting point and g the goal.

In our implementation, we used N=20 weights per each DoF and different time scaling factors, which summed up to having 67 input/output parameters of the AE neural network. The neural network was fully connected with [67 15 3 15 67] neurons in each layer and all the values were normalized in range [-1,1]. Once trained with source environment database, this network is not changed any more. With this network, we transform our initial dataset given in Eq. (4.1) to the dataset for initial training of the smaller neural network:

$$\hat{\mathcal{D}}_{\text{so}} = \left\{\boldsymbol{V}^l, \boldsymbol{T}\right\}. \tag{4.9}$$

Therefore, our network is capable of connecting the targets to the latent space of the AE:

$$\boldsymbol{T} \mapsto \boldsymbol{V}^l. \tag{4.10}$$

The main focus is put on the smaller network shown in yellow in Figure 4.4. In the beginning, it is trained only with the dataset of kinematic trajectories, but its weights will be later updated with the knowledge from the target environment using backpropagation and RL. This update is marked with an orange circle in Figure 4.4.

Complete system for approximating the required trajectory works in such a way that the small network receives the information about the required target, connects it to latent space of AE network and afterwards the decoder part approximates the CDMP parameters that are decoded and the final trajectory is sent to the robot.

Throwing with humanoid robot

For the case of throwing with a humanoid robot, we used a more straightforward approach of connecting the landing spot directly with CDMP parameters. However, since the trajectory was hard to fit into the workspace, we did not change the initial and goal position, but only weights $\boldsymbol{w}_i^p$ and $\boldsymbol{w}_i^o$ that were corresponding to translational movements in x and z axes and rotational movement about y axis as well as the time of trajectory execution τ. We used $N = 25$ for the number of CDMP weights in this case, and it sets the output of our NN to 76 parameters:

$$\boldsymbol{V} = \left[\boldsymbol{w}_{\{x\},i}^T, \boldsymbol{w}_{\{z\},i}^T, \boldsymbol{w}_{\{rot_y\},i}^T, \tau\right]^T. \tag{4.11}$$

We used a fully-connected network and all the inputs/outputs were normalized values within a range of $[-1, 1]$ and layer sizes $[2, 15, 30, 40, 70, 76]$. Activation function used for this network was *tanh* and it was capable of mapping:

$$N_{\text{so}}(\mathcal{D}_{\text{so}}) : \boldsymbol{T} \rightarrow \boldsymbol{V}. \tag{4.12}$$

Since the other parameters of CDMP are fixed we can consider that this network provides us with an end-to-end mapping:

$$N_{\text{so}}(\mathcal{D}_{\text{so}}) : \boldsymbol{T} \rightarrow \mathbf{W}. \tag{4.13}$$

4.1.3 Adaptation of knowledge to target environment

When the source environment knowledge is finally stored in NNs, we switch to the target environment where it does not perform that well anymore because of the difference in modelling. We update the NNs with the newly acquired data.

Throwing with robot arm

In the first case, the autoencoder is trained on a huge amount of synthetic data, synthesized in a manner that covers most expected robot motions. Note that the autoencoder works on CDMP parameter data, and thus does not need any real-world execution data. Therefore, we are now only left with the second network, which maps the target coordinates to the latent-space parameters of a deep autoencoder $\boldsymbol{T} \mapsto \boldsymbol{V}^l$. Again, a dedicated DNN can be used for this mapping. Accelerated learning of the dedicated DNN (presented in yellow in Figure 4.4) for mapping between the throwing target coordinates and the deep autoencoder latent space parameters is our main focus.

Algorithm 4.1: Procedure of improving neural network connecting targets with the latent space values.

procedure Improve NN
  set targets $\mathbf{T} = \{x_z, z_z\}_{z=1}^{Z}$
  set $\boldsymbol{\theta}_1^{RL}$ based on the difference between two NNs
  set $i = 0$
  repeat
    increase i
    add noise on RL parameters $\boldsymbol{\theta}_i^{RL} = \boldsymbol{\theta}_i^{RL} + \boldsymbol{\epsilon}$
    set corresponding NN parameters to $\boldsymbol{\theta}_i^{RL}$
    set R=0
    for z=1:Z
      use NN to map: $\boldsymbol{T}_z$ -> $\boldsymbol{\theta}_m^l$
      use decoder to map: $\boldsymbol{\theta}_z^l$ -> $\boldsymbol{\theta}_z$
      execute motion in the target domain
      measure landing spot $\boldsymbol{T'}_z = [x'_z, z'_z]$
      give reward $\hat{r}_z = ||\boldsymbol{T}_z - \boldsymbol{T'}_z||$
      $R = R + \hat{r}_z/Z$
    end
    train NN with $\boldsymbol{T'}_z$ -> $\boldsymbol{\theta}_z^l$, z=1...Z;
    use RWPL to update $\boldsymbol{\theta}_{i+1}^{RL} = f(\boldsymbol{\theta}^{RL}, R)$
    reduce search noise $\boldsymbol{\epsilon}$
  until i =number of updates

The first step in our algorithm is to check which weights and biases change the most when changing the domain, i. e., when going to dynamical simulation. For this purpose, we execute several (we used 9) throws in dynamical simulation and measure the landing spot. These can be throws to random targets, or they can be evenly distributed over the target area. The latter is less likely to over-fit to one random area, but it requires more knowledge and input by the user. With this small amount of data, we retrain the dedicated network in only several epochs (we used 10), and check the difference in the weights and biases. We sort the weights and biases by the difference and use several ones (we used $I = 15$ of both) that change the most to be modified by the RL algorithm:

$$\boldsymbol{\theta} = \{\hat{\omega}_r, \hat{b}_r\}_{r=i}^{I}. \tag{4.14}$$

Since these parameters are changed by both the RL and retraining, and the others only by retraining, it is very similar as freezing the other parameters.

In the next step of transfer learning, we choose queries that are equally spread over the robot workspace. For this, we have chosen Z equally spaced targets (we used separately 12 and 16) in the same range as in our kinematic database. In order to improve performance of the dedicated network, we learned to throw simultaneously on all Z targets and we were assigning them a cumulative reward. After executing throws at the targets, we retrained the network in only one epoch by using newly obtained data, and updated $\boldsymbol{\theta}_{RL}$ with RWPL (see Sec. 2.1.5). For clarity, the procedure describing the learning steps is given in Algorithm 4.1.

Throwing with humanoid robot

In the second setup, our source domain was simulation in Gazebo environment and our target domain simulation in MuJoCo. After training the initial target-to-motion NN model of the throwing skill in the source domain, we adapted the model using different approaches:

- Backpropagation (BP)

  When using this approach, initially (4.12) is used as the skill model for target domain actions. But, after a certain amount of data is gathered in the target domain, $\mathcal{D}_{\text{ta}} = \left\{\mathbf{W}_{m,\text{ta}}, \boldsymbol{q}_{m,\text{ta}}\right\}_{m=1}^{M'}$, where subscript ta stands for target, this network gets retrained using backpropagation

  $$N_{\text{BP}}(\mathcal{D}_{\text{ta}}) : \boldsymbol{T} \rightarrow \mathbf{W}. \tag{4.15}$$

  The training is done on all the layers (none were frozen), and using all target domain data so that only initial parameters of the network are affected with training from the source domain.

- Reinforcement Learning (RL)

  When using Reinforcement Learning (RL) approach, all but one of the NN layers are frozen. The weights and biases of this layer are updated using a RL algorithm, which tries to maximize the reward. The reward needs to be predefined. For the throwing example case, it can be based on the smallest distance of the ball-flight trajectory from the desired target. To obtain the same precision in the whole throwing range, it needs to be applied on a set of equally spread targets and accumulated over all the throws,

  $$N_{\text{RL}}(\text{RL}(N'_{so})) : \boldsymbol{T} \rightarrow \mathbf{W}. \tag{4.16}$$

  Here N', means that only one layer was changed. For RL algorithm, we used RWPL which is described in detail in Sec. 2.1.3

- Combined RL&BP

  Combined RL and BP approach is a mixture of aforementioned approaches. All the layers but one were frozen and parameters of the layer were updated first with RL algorithm and later this layer of the network was trained with backpropagation but only in a small number of epochs. The training and reward updates were done on equally spread targets.

  $$N_{\text{RL\&BP}}(\text{RL}(N'_{ta}(\mathcal{D}_{\text{ta}}))) : \boldsymbol{T} \rightarrow \mathbf{W}. \tag{4.17}$$

4.2 Experimental Evaluation and Results

Here, we describe the experimental setups used for the both experiments. After providing explanations of the source and target environments used in both cases, we proceed with obtained results.

4.2.1 Experimental setup

Throwing with robot arm

For the experiment involving throwing with the robot arm, the source environment knowledge consisted of pre-calculated trajectories. In addition, we developed a kinematic simulation in MATLAB that took into account the dimensions of the Mitsubishi PA-10 robot (Figure 4.5-left). During the throwing action, only three joints of the robot arm were actively controlled. It is worth noting that this setup does not compromise the generality of the approach, as the robot arms can be easily repositioned towards the target by manipulating the first (bottom) joint. To perform transfer learning, we employed the MuJoCo Haptix simulation controlled through MATLAB. Similar to the kinematic simulation, we utilized 7 DoF Mitsubishi PA-10 robot with a throwing spoon and with only 3 joints active while the others were fixed. This ensured consistency between the source and target environments, as well as reduced the complexity of the TL process. The target environment setup is depicted in Figure 4.5-right.

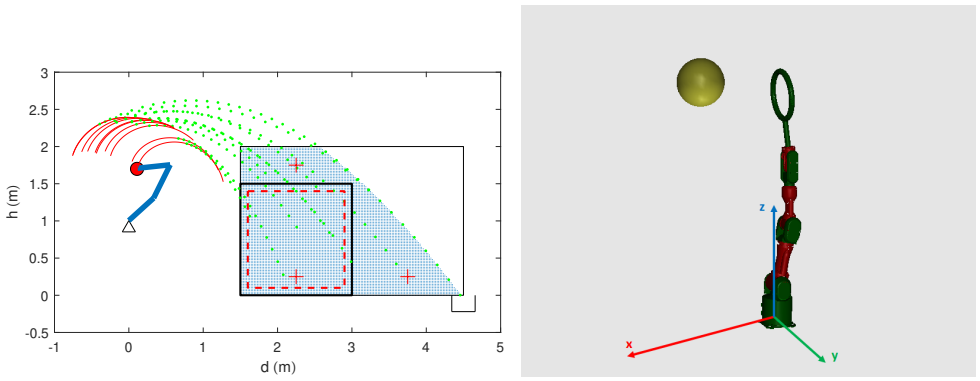


Figure 4.5: Experimental setup showing Mitsubishi PA10 robot in two domains: Kinematic model and MuJoCo simulation.

Throwing with humanoid robot

For the case of using the humanoid robot TALOS for throwing, we used inaccurate Gazebo simulation as the source environment [121] and MuJoCo Haptix simulation connected to MATLAB as the target environment. The target is also always in the sagittal plane of the robot, but rotated for 3° towards the right side. This allowed for the maximal exploitation of the relatively constrained workspace of the TALOS humanoid robot [62]. The throwing action is always an over-the-head throw forward with the robot's left arm. The throwing action trajectories are provided to the robot in the task space, where 6 DoF motion in space is defined. Because the arm of the TALOS has 7 DoF, it is functionally redundant for the given task. An inverse kinematics solution is calculated with a null-space controller, where the secondary task is to always keep all the joints as far from joint limits as possible. The throwing action did not influence robot stability and thus, no controllers for the maintaining of the postural stability were used.

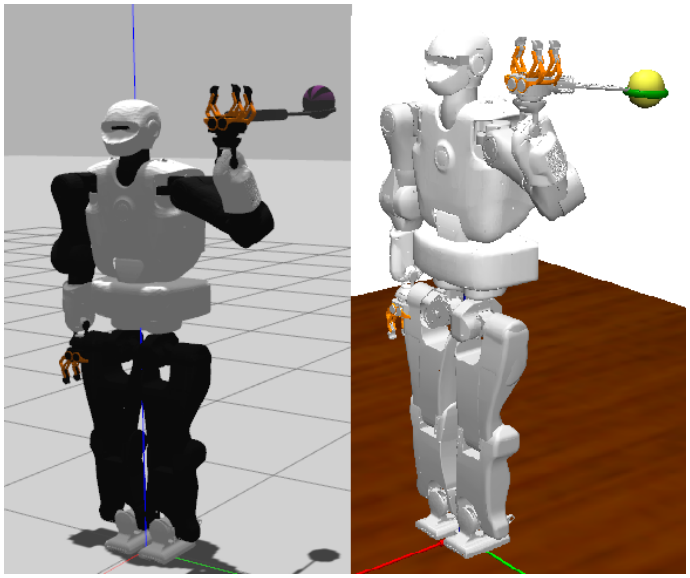


Figure 4.6: Experimental setup showing throwing action in two different domains: Gazebo and MuJoCo simulation.

4.2.2 Transfer learning evaluation

Evaluation of throwing with robot arm

In order to evaluate our approach, we compared our method to a simple backpropagation method. After preparing the database of executable trajectories containing 3247 targets and corresponding trajectories, all the trajectories were encoded as CDMPs and used first for training of the autoencoder network that contained 3 latent space values. A dedicated network containing [2 3 15 5 3] neurons in its layers was trained to match these targets to corresponding latent space values as given by (4.10).

We repeated two sets of experiments with different amounts of targets for learning. In the first set, we learned Z=16 targets equally spread within the range of the robot. In the second set of experiments, we repeated with Z=12 targets. We learned all Z targets in 9 iterations, which sums the learning procedure to 144 shots for the case of learning 16 targets and 108 shots for the case when learning 12 targets. After each iteration of learning, the performance of the NN was tested on 64 targets shown with red circles in Figures 4.8-4.10. For backpropagation, we used 200 randomly chosen targets. After each shot, NN was updated with the new data in 10 epochs and the performance was measured on the same 64 targets as in the case of learning with the method given in Alg. 4.1.

Figure 4.7 shows the convergence of the error when doing transfer learning using 16 targets, 12 targets and when using backpropagation. Error (and reward) is given by

$$\hat{r}_z = ||\boldsymbol{T}_z - \boldsymbol{T'}_z||, \tag{4.18}$$

where, just as in Alg. 2.1, $\boldsymbol{T}_z$ is the target of z-th throw and $\boldsymbol{T'}_z$ is the landing spot of the same throw. Data for the case of learning 16 targets is given in red, for 12 targets in green and for the case of BP in blue. Upper graphs show the average error convergence and lower graphs show the maximal error convergence. We can see that the proposed algorithm evidently outperformed BP in the number of required executions and achieved accuracy in both cases when using 12 and 16 targets. Although the top plot shows that the average error of throwing was even better for 12 targets, the maximal error was lowest when using our approach and 16 targets.

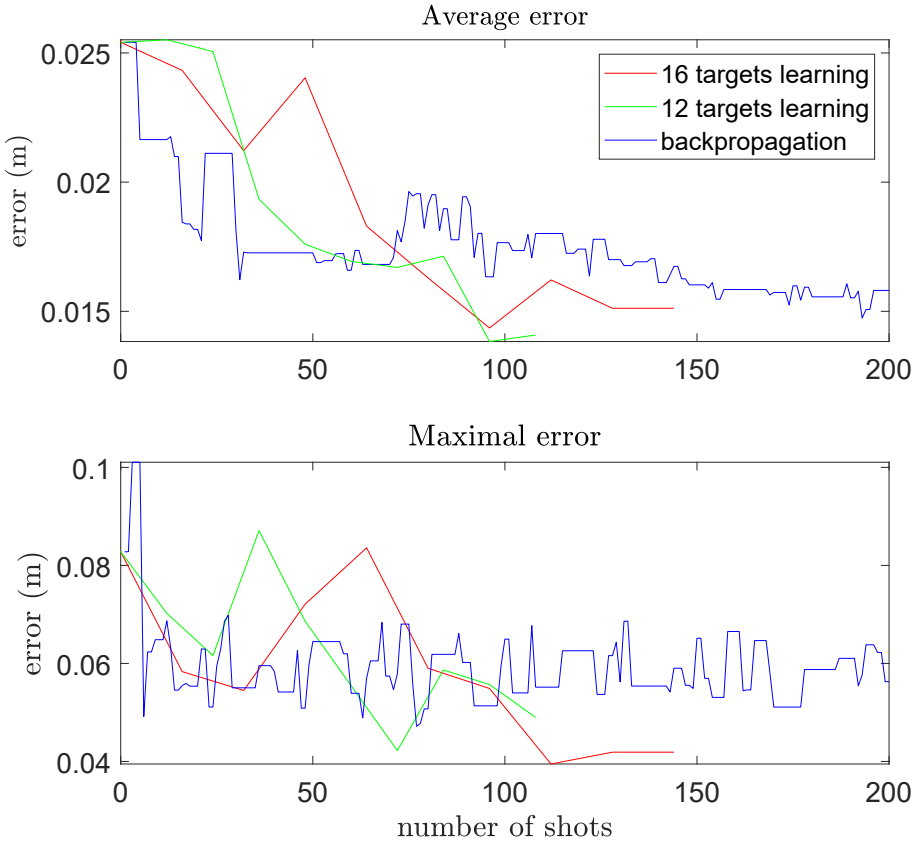


Figure 4.7: Convergence of the error when we used our method on 16 targets is shown with a red line, with 12 targets with a green line and with backpropagation with a blue line.

Results in Figure 4.8, presented as a contour plot, show an error of throwing in the new domain (dynamic simulation) trained only using the initial kinematic database.

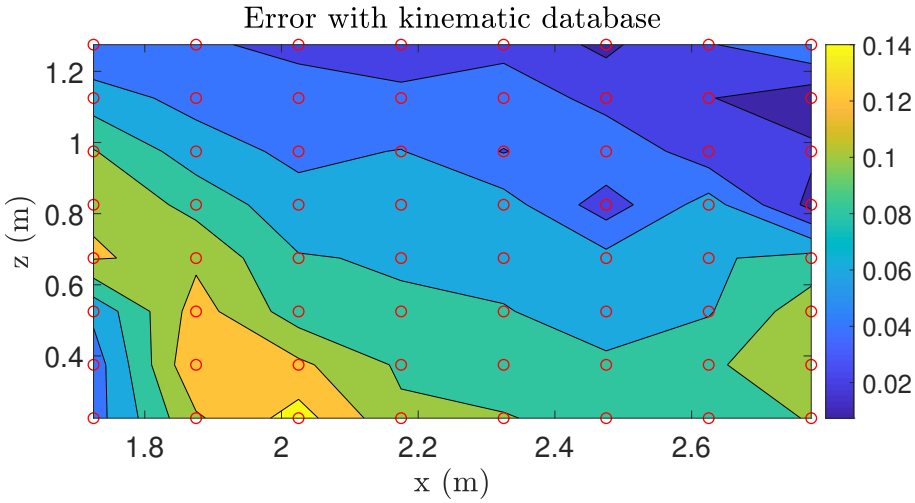


Figure 4.8: Contour plot of the error with using the kinematic database for training the neural network.

Figure 4.9 illustrates the error observed after the first update, which occurred after updating the NN with 9 random targets. These targets were used to determine important

parameters for RL. Therefore, this plot demonstrates the performance of the NN at the initial stage of learning using our proposed approach.

From the plot, it is evident that the adaptation of the action commences rapidly. In just nine shots, the maximum error decreased from approximately 14 cm to around 8 cm. These results indicate promising initial convergence when employing a simple backpropagation (BP) approach.

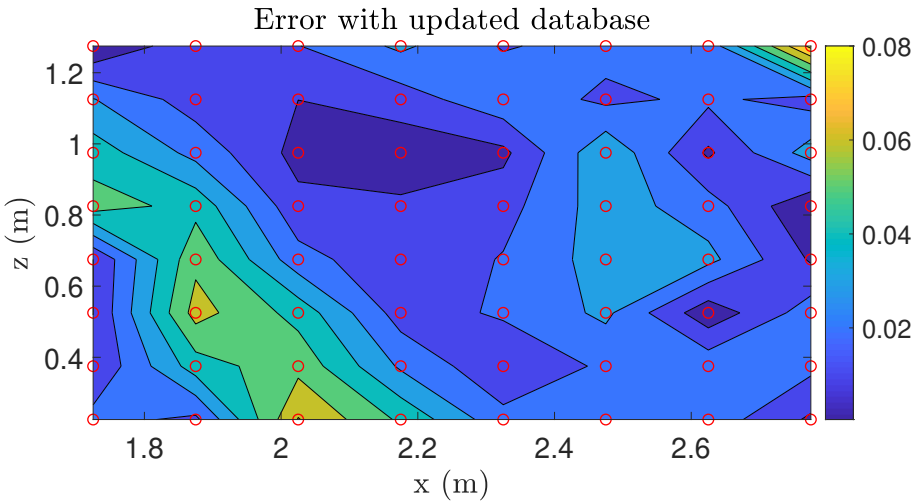


Figure 4.9: Contour plot of the error after the update of the neural network with only 9, equally spaced targets. This also represents the initial point for our proposed approach.

Contour plot in Figure 4.10 shows the performance of the network at the end of the training with our approach when using 16 targets. It shows that the maximal error dropped below 4 cm at the end of the learning, while the average error was below 2 cm.

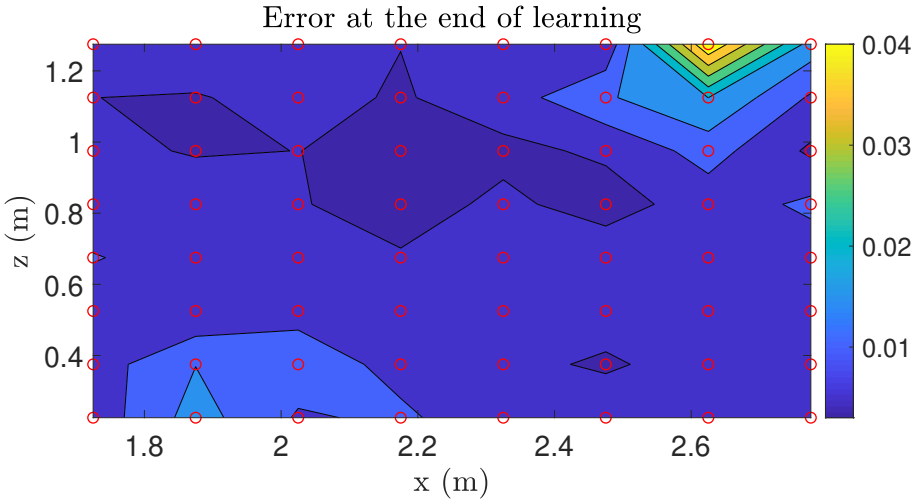


Figure 4.10: Contour plot of the error at the end of learning.

Bar plots in Figure 4.11 sum up the statistics for the average and maximal error in the case of directly using the kinematic database, at the beginning of the learning and at the end of the learning for the case where 16 targets were used.

The results show an obvious decrease in error at each step of the learning process. Execution of only 9 targets and updating the network gives a 64.5% decrease in average

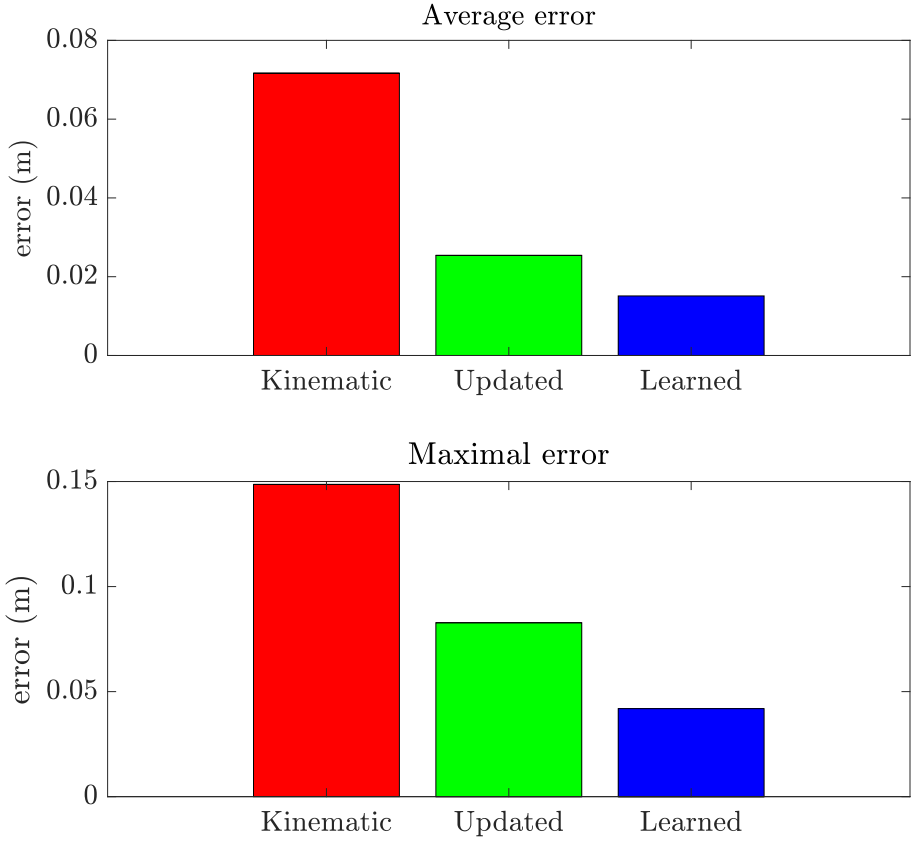


Figure 4.11: Bar graphs of the mean and maximal error for the case of kinematic database, after the first update and at the end of the learning.

error. However, as it can be noticed from Figure 4.7, this backpropagation method comes to a saturation point after approximately 100 iterations. On the other hand, a major benefit of our method, where retraining of the network is combined with performing RL, is that it constantly continued to improve the performance. After 144 iterations it gained an additional 40.5% drop in the error.

Chaining of DNNs can be applied, for example, to improve classification [122]. In our case, using a deep AE and its decoder, which is chained to the network capable of connecting the target to CDMP parameters, has a profound impact on the effectiveness of the approach. Because the dimensionality of the task is greatly reduced, a lower number of "real-world" trials and RL executions is needed. Figure 4.12 shows the error of throwing when the kinematic database is applied directly to CDMP parameters, given with (4.7) and (4.8). As the results show, the error of throwing in the adapted domain (dynamic simulation) is in the range of 1 m, so 50 times higher than when the dimensionality is reduced. This is caused by the fact that AE, due to its extraction of only the important parameters, behaves as a filter. When using directly CDMP parameters, even a small error in one of the weights can cause a jerky trajectory that is not capable of performing the task. Even when the throwing NN is updated with the presented approach of varying only some of the most important parameters, the error remains in the range of half a meter and more. On the other hand, when using the dimensionally reduced data, as in our approach, the error is in the range of a few cm. Table 4.1 sums up the obtained results.

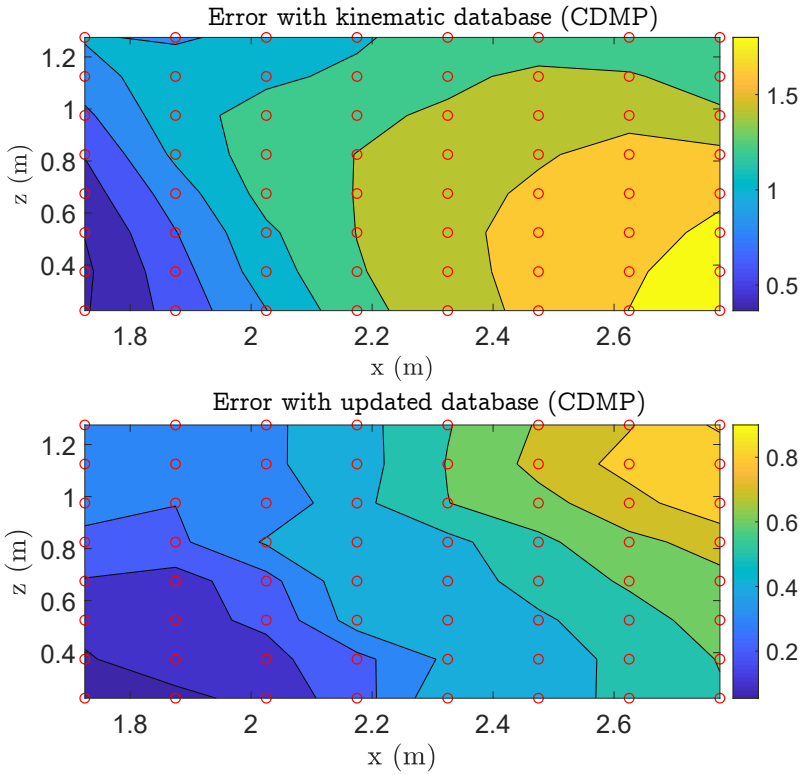


Figure 4.12: Contour plot of the error with kinematic and updated database when using CDMP instead of latent space.

	AE_k	AE_{upd}	$AE_{upd} + RL$	$CDMP_k$	$CDMP_k$
Avg	0.072	0.025	**0.015**	1.293	0.918
Max	0.148	0.082	**0.041**	1.903	0.453
NS	3247	3247+9	3247+9+144	3247	3247+9

Table 4.1: Average, maximal error and number of samples used for training of NN. Samples obtained in a new domain (dynamic simulation) are underlined.

Evaluation of throwing with humanoid robot

We tested our proposed few-shot transfer learning approach and compared it to other approaches in the previously presented throwing task example. The following list describes the different experiments. Note that the same names for learning approaches are used in Figures 4.14 – 4.16.

- **Gazebo:** Action execution in the initial simulation domain (Gazebo) using the original target-to-motion NN model from the source domain using Eq. (4.12).
- **Initial:** Direct application of the original target-to-motion NN model, given with (4.12) from the source domain (Gazebo) without any modification, but in the target domain (MuJoCo).
- **RL+BP:** Modified target-to-motion NN model in the target domain (MuJoCo) using Eq. (4.17). The targets for throwing are predefined and equally spread.

- **RL+BP-r:** Modified target-to-motion NN model in the target domain (MuJoCo) using Eq. (4.17) but the targets for throwing are randomly chosen within the throwing range.
- **BP-r:** Modified target-to-motion NN model in the target domain (MuJoCo) using Eq. (4.15), but the targets for throwing are randomly chosen within the throwing range.
- **RL:** Modified target-to-motion NN model in the target domain (MuJoCo) using Eq. (4.16). The targets for throwing are predefined and equally spread.
- **BP:** Modified target-to-motion NN model in the target domain (MuJoCo) using Eq. (4.15), but the targets for throwing are predefined and equally spread.

The first step in all the approaches was to get the database and to train the NN skill model in the source domain. This was done in a highly inaccurate Gazebo simulation (poor contact modelling and time-step synchronization when connecting to the ROS network). This way, we acquired almost 10000 executable trajectories. After removing the trajectories that caused the ball to drop of, hit the robot and or where ball flight was unrealistically long, we were left with $U = 3000$ examples to train the NN with layer sizes $[2, 15, 30, 40, 70, 76]$. The database of acquired trajectories is given in Figure 4.13.

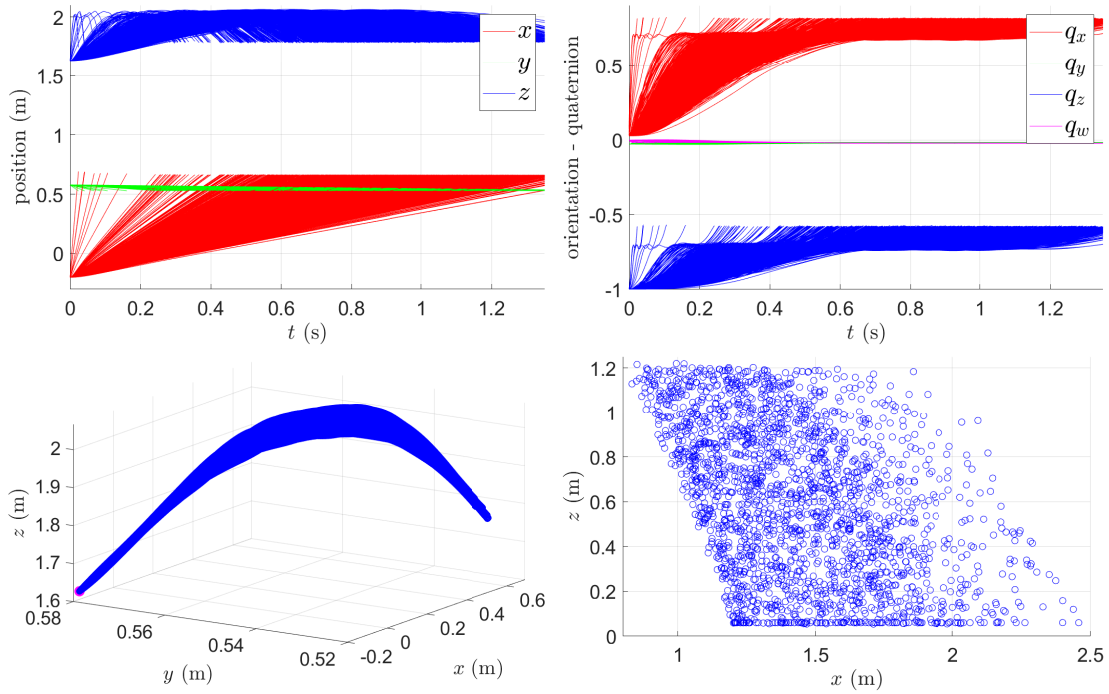


Figure 4.13: Database for training the motion-to-target NN skill model of throwing, acquired in the source domain. From left to right: Position trajectories in Cartesian space; Quaternion orientation trajectories; spatial trajectories with magenta color marking the start; targets, i. e., landing spots.

In order to test how well this NN skill model performs in its domain (Gazebo), we performed the **Gazebo** experiment and tested it on 25 equally spread targets: $\boldsymbol{T}_t = [x_t, z_t]$. They were chosen so that the distance from the robot (x-axis) is in the range of $x_t \in [1.1, 2.3]$ m and height of target (z-axis) is in the range $z_t \in [0, 1.1]$ m.

The protocol for the **initial** experiment was the same: 25 throws on equally spread targets, but in the more stable MuJoCo Haptix simulation that was used as our target environment.

The protocols for the remaining experiments are outlined as follows. In these experiments, the throw was executed towards a set of 25 targets that were evenly distributed. The targets ($\boldsymbol{T}_r$) were positioned such that the x-axis coordinates ranged from 1 to 2.4, while the z-axis coordinates ranged from 0 to 1.2 m. After collecting data on the landing spots of all the r-targets, we proceeded to update the network using one of the proposed methods. Following the update, we evaluated the quality of the learned model by testing it on a separate set of test targets.

For the **BP-r** approach, the robot performed throws towards 25 randomly selected targets. The resulting landing spots were recorded, and this data was used to update the network. The precision was tested on 25 test targets.

In **RL** experiments, we have chosen the hidden layer with the smallest number of parameters to be retrained, as high dimensionality of the search space may make RL too slow. We were retraining the second layer of the NN skill model, which had 15 nodes. This layer has only two inputs so we retrained 2×15 weights and 15 biases. This led us to searching in only 45-dimensional space. Regarding the reward mechanism used in the experiments, each r-target (target location) was evaluated separately, and a reward value was assigned to it. The final reward for each learning iteration was calculated as the sum of these individual reward values. The reward r_i for each r-target was determined using the following equation

$$\begin{aligned} R &= \sum_{i=1}^{25} r_i, \\ r_i &= 1 - \|\boldsymbol{T}_{r,i} - \boldsymbol{T'}_i\|/2, \end{aligned} \tag{4.19}$$

where $\mathbf{T}_{r,i}$ represents the coordinates of the reward target, and $\mathbf{T'}_i$ represents the coordinates of the actual hit achieved during the throwing action. The reward calculation was based on the Euclidean distance between the reward target and the actual hit, scaled by a factor of 2.

In **RL+BP**, the reward was given in the same manner, but the retraining of the network was also added after executing throws at $\boldsymbol{T}_r$. In this case, retraining was done in only 10 epochs so that it does not totally cancel the parameters found by the RWPL RL algorithm.

For **RL+BP-r**, everything was done in the same manner as in **RL+BP**, just the targets were randomly chosen within the range.

Figure 4.14 shows the average throwing error when transferring to the target domain. All approaches reduce the average error after several test iterations. Note that one test iteration includes 25 throws (additional 25 were preformed for the evaluation of the learning after each iteration). The proposed RL+BP method is the fastest of the tested methods. Also BP-r method and RL+BP-r are relatively fast. On the other hand, BP method consistently diverges after some time. This is because the training targets are predefined and therefore hits are similar, thus there is not enough diversity in the data. Consequently, the approach overfits. Qualitatively similar results, but for the maximal error of throwing, are shown in Figure 4.15, where RL+BP method is fastest, while BP method diverges after some time.

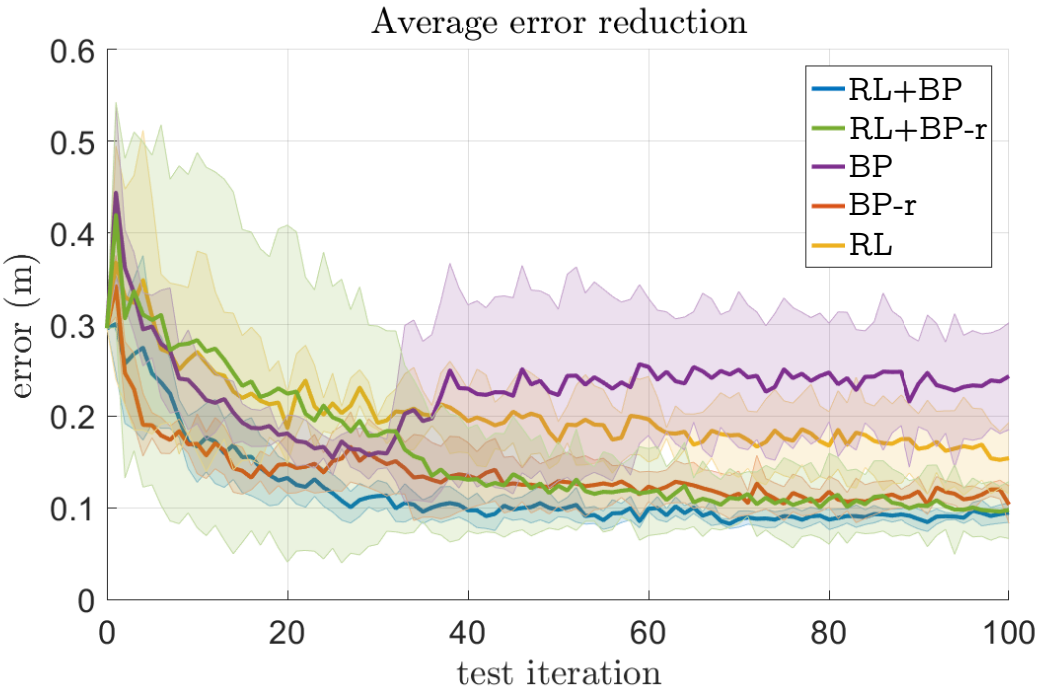


Figure 4.14: Reduction of the average (left) and maximal (right) throwing error after transitioning to the new domain (sim-to-sim) using different methods. Solid lines show the average of 5 experiments while the shaded area depicts the standard deviation.

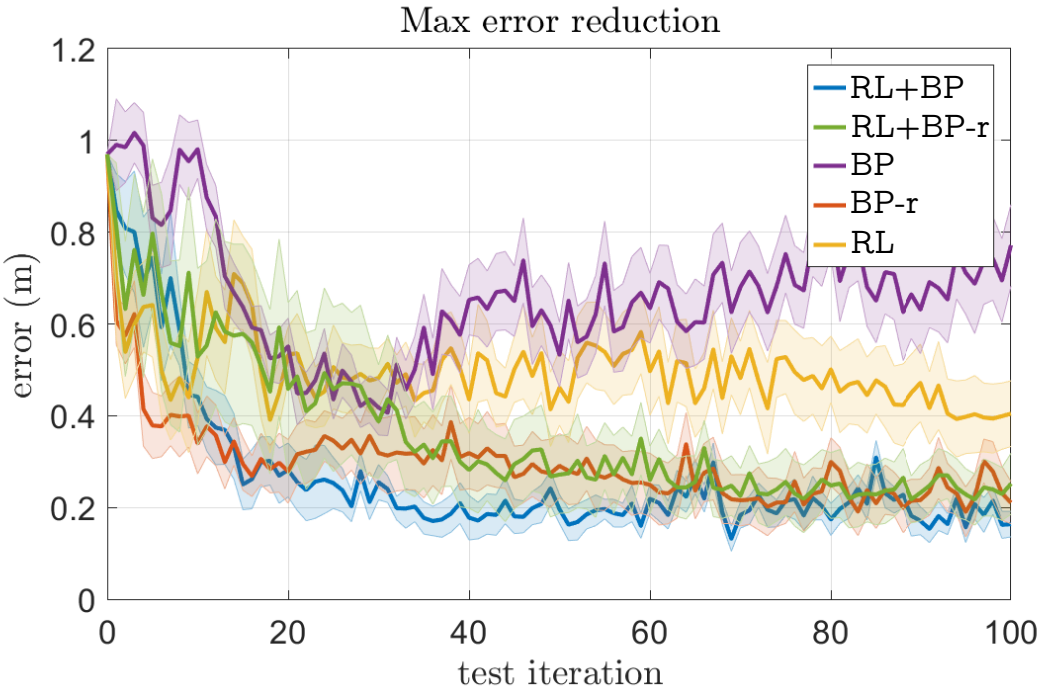


Figure 4.15: Reduction of maximal throwing error after transitioning to the new domain (sim-to-sim) with different methods. Solid line shows the average of 5 learning iterations and shaded area depicts standard deviation.

Figure 4.16 presents the accuracy of the throwing action within the throwing range area. It is important to note the varying scales on the contour plots, indicating the different levels of error. By analyzing the contour plots, it becomes evident that the RL+BP approach outperforms the other methods in terms of accuracy, providing valuable insights into the effectiveness of this combined approach in optimizing the throwing trajectory towards the desired targets.

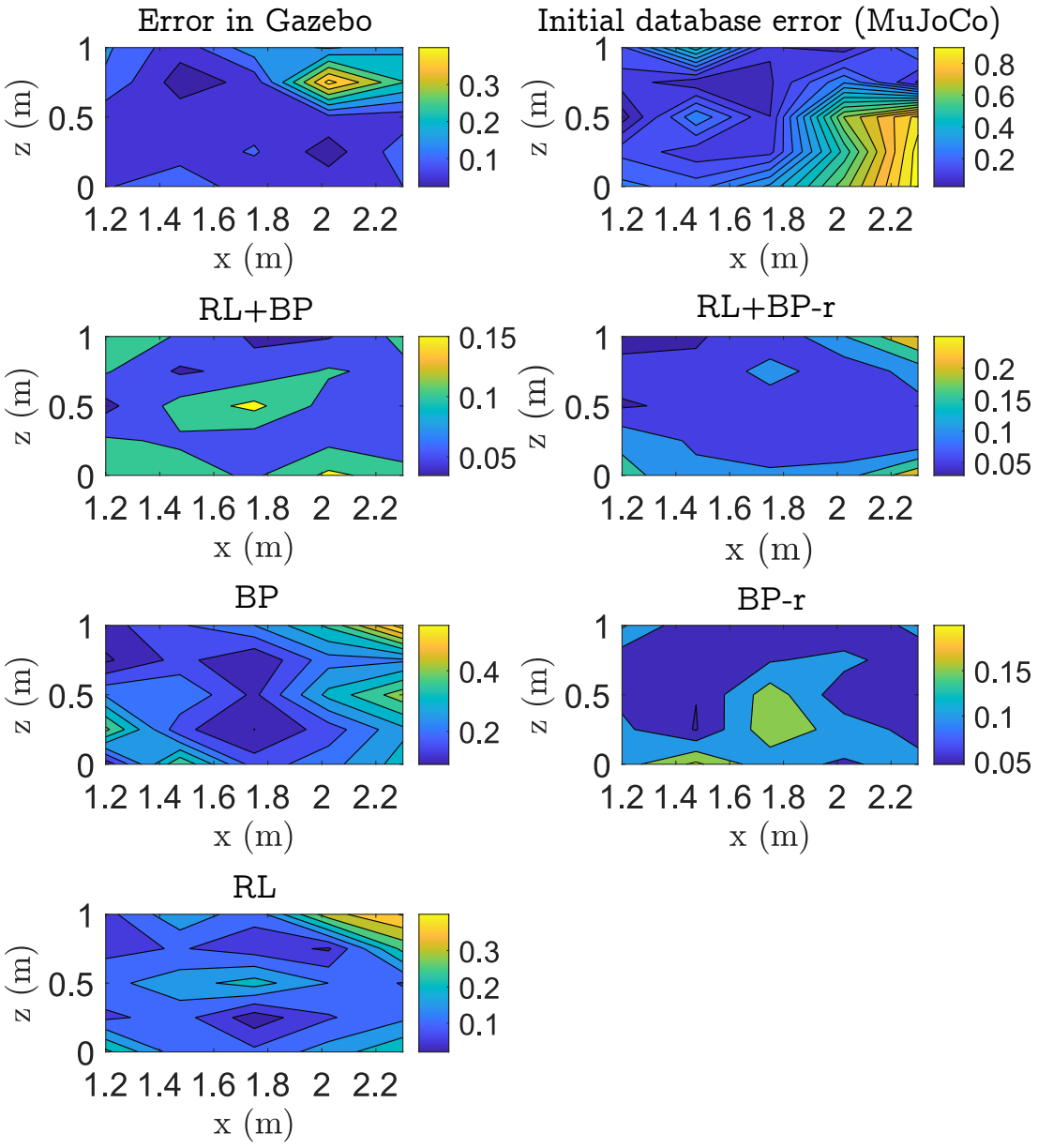


Figure 4.16: Contour plots showing the throwing error after learning for different learning approaches.

Figure 4.17 shows the averages of errors after the end of learning. The initial target-to-motion model, when directly transferred, resulted in considerably larger error, but after roughly 30 learning iterations, the RL+BP method matched the average accuracy, and performed much better in maximal error. The advantages over using only BP, RL and BP-r are clearly shown, as well as the divergent result of the BP method.

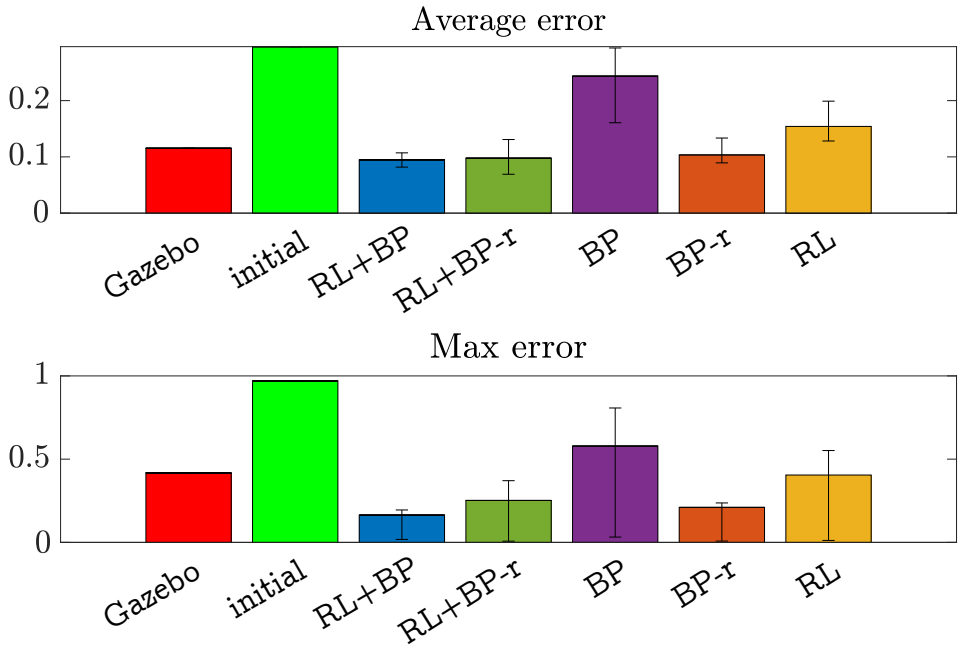


Figure 4.17: Average error (top) and maximal error (bottom) at the end of learning (average of 5 learning trials). Error bars depict the minimum and maximum obtained values.

In our experiments, we performed 25 throws in one iteration, however, less could be applied. Results in Figure 4.18 show the results where only 5 equally spread throws were applied for one iteration. Evaluation was still performed with data of 25 throws, so that the results are comparable with the previous. However, again, these were not used for the learning. While these results are also in simulation, the numbers of learning throws are more realistic for real-world implementation.

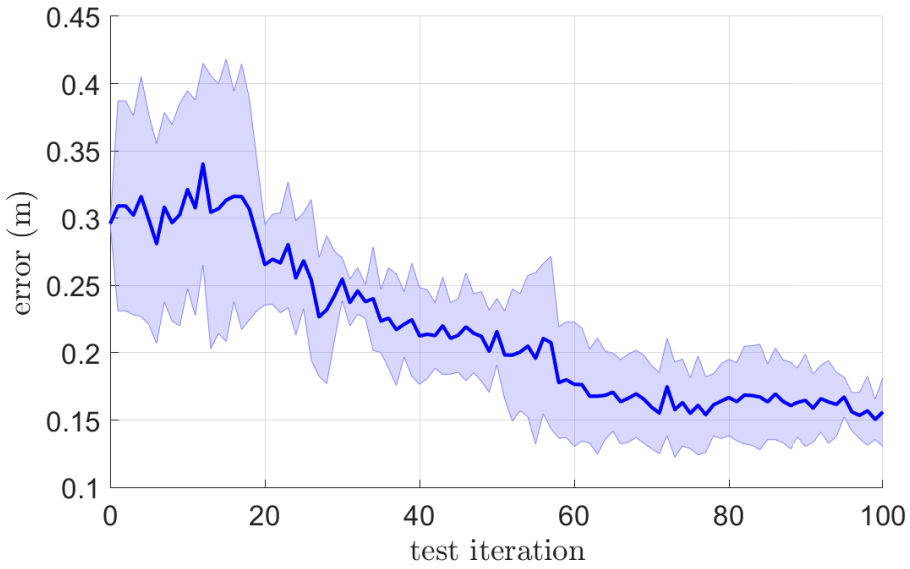


Figure 4.18: Average error reduction when only 5 throws were implemented for one learning iteration in **RL+BP** setting.

4.3 Probabilistic Approaches for Storing the Knowledge

Previously presented methods were storing the knowledge in the networks that were trained to connect one target to one set of parameters. However, it has been recently shown that sometimes, non-deterministic imitation learning methods that are not trained to always output the same solution can exhibit slightly better transferability [57]. This can be useful when there are more simulation parameters that influence the behaviour. This is why our approach was tested on a more complex problem of a robot manipulating a box by dragging it on the table using only a rod as an end-effector [63]. Since the task was performed in the task space of the robot, in order to avoid problems with calculation of the inverse kinematics, we used a redundant Franka Panda robot with 7-DoF. The presented task is particularly hard even for a human because it requires a lot of patience and corrections in order to achieve the desired box position and orientation. Our setup is shown in Figure 4.19-right. The box was initialized in a random position (shown in red) and the goal for the robot was to move the box to the desired and always the same final position (shown in green).

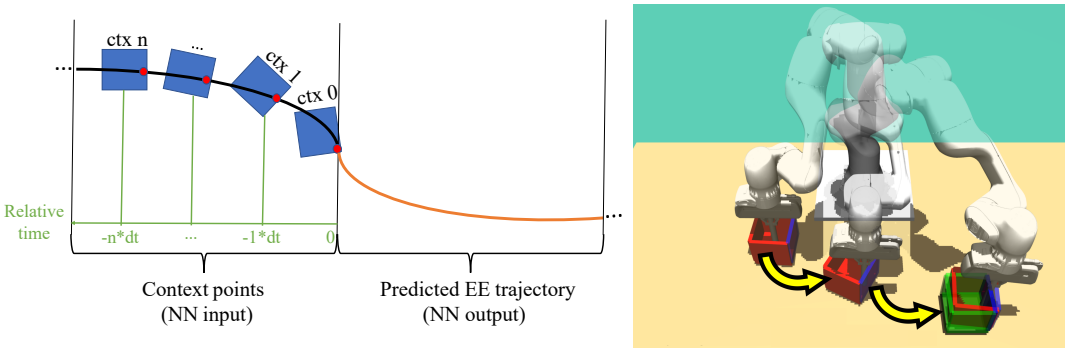


Figure 4.19: Left: Inputs and outputs of the trajectory prediction architecture: Input to the network are several context points. Each context point contains information about box position and orientation (blue) and information about robot end-effector position (red) together with relative time in which it happened (green axis). Output of the architecture is the predicted end-effector trajectory (orange). Right: Execution of each replanned part of trajectory on a simulated robot.

We still explore a deep neural network-based encoder-decoder motion architecture that generalizes well between task instances, but with putting our main accent on architectures with a potential in enabling the seamless transfer to a new domain without adding any additional information from the user (like in Sec. 4.1.3 where we manually selected targets for reward and evaluation). Similar to throwing, this is also a dynamic task, where the final position of the robot is not directly related to the successful outcome of the task. We tested two different architectures. Since the approaches are probabilistic (on the same requested target positions, they are capable of predicting the whole distribution of the solutions instead of only one), we had to combine them with probabilistic trajectory representations. The approach has replanning steps during the execution, and therefore it requires ability for smooth transitions in-between the predicted trajectory segments. This would be hard to accomplish with ProMPs, so we combined the motion generation with a novel, ProDMP encoding methodology [56]. Except for enabling us for smooth transitions, it is much faster than regular DMPs in its integration step [56], thus the forward pass through the network is also faster when approximating the trajectory. It should be noted that these movement primitives are a good choice for this task, but they are not capable of decoupling the

velocity profile from the trajectory and could not be used for the optimization processes as presented in Chapter 2. Detailed information including auxiliary math describing the ProDMPs is given in [56].

The applicability of the proposed approach depends on the size of the latent space, the means of sampling and the level of aggregation of data before predicting the behaviour of the executed dynamic task. The transferability, on the other hand, depends on the diversity of data. Below, we explore these aspects of motion generation that should maintain the correspondence of a dynamic task.

4.3.1 Methodology

The database for training the neural network was acquired by human demonstration where the person was kinaesthetically guiding a simulated robot in order to push the box from the randomly initialized position to the final position.

Two different architectures presented in this section were tested with two different ways of choosing the action. The action could be chosen from the distribution by only randomly sampling from the distribution or by taking the mean of sampled parameterized actions. With both architectures, the input of the neural network was the current state of the box (position and orientation) together with the current state of the robot (position of the end-effector). Since the task is complex, we also introduce the so-called aggregation where we do not only take the current state in consideration when choosing the next action, but also take into consideration multiple previous robot/box states. In addition to this, we also briefly experiment with determining the required latent space size of the network.

The goal of both architectures is to predict the motion trajectory based on the state of the setup. The state of the setup can be described using information about current box position in x and y axis and orientation about z axis (denoted with α_{box}) as well as robot end-effector position and velocity in x and y axis. It should be noted that z position of the end-effector is held constant and its shape was cylindrical so that its orientation does not have an influence on the movement of the box. For this reason, those two values were left out. Together with the information about the relative time at which the system was in the given state, we obtain the necessary information for predicting the trajectory - Figure 4.19 (left). This information is called context and each context point contains eight values: $\mathbf{ctx} = [x_e, y_e, \dot{x}_e, \dot{y}_e, x_{box}, y_{box}, \alpha_{box}, t]$.

As shown in depictions of the architectures in Figures 4.20 and 4.21, several (m) context points are first sent to two encoders (μ and σ^2) in order to get the latent space observation ($\boldsymbol{\mu}_r$) and an uncertainty $\boldsymbol{\sigma}_r^2$ measuring how important this observation is. The following step is to obtain latent state posterior $p(\boldsymbol{z}|\mathbf{ctx})$ which is given by a factorized multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z^2)$. For this purpose, aggregator is used. As an input, it takes the set of latent observations and their uncertainties together with the prior distribution: $(\boldsymbol{\mu}_A, \boldsymbol{\sigma}_A^2)$. Prior distribution was set to be normalized Gaussian distribution with zero mean. For both architectures, this part is the same. The difference comes from the part where the trajectory representation is sampled and decoded.

The first architecture that we used included Parameter-Based (PB) Decoder [123]. This decoder consists of two decoders that take the posterior distribution as an input and output mean value of the ProDMP weights and goal ($\boldsymbol{\mu}_{w_g}$) as well as Cholesky decomposition of the covariance matrix of the weights and goal distribution ($\boldsymbol{L}_{w_g}$). This is further used by ProDMP architecture that is, when given initial position and velocity of the end-effector $(\boldsymbol{y}, \dot{\boldsymbol{y}})$, able to convert these values into the mean and covariance of the trajectories: $\boldsymbol{\mu}_\tau$, $\boldsymbol{\Sigma}_\tau$. For training of this network, negative log-likelihood cost function that increases the possibility of observing the trajectory $\boldsymbol{y}$, given the distribution with mean $\boldsymbol{\mu}_\tau$ and covari-

ance $\boldsymbol{\Sigma}_\tau$ that are determined with decoder NN parameters $\boldsymbol{\theta}$, was used:

$$\mathcal{C}(\boldsymbol{\tau}, \boldsymbol{y}_b, \dot{\boldsymbol{y}}_b, \mathbf{ctx}) = -\log \mathcal{N}(\boldsymbol{y}|\boldsymbol{\mu}_\tau, \boldsymbol{\Sigma}_\tau, \boldsymbol{\theta}). \tag{4.20}$$

An important thing to note with this architecture is that the trajectory is sampled from the distribution of ProDMP parameters when executing the forward pass in the evaluation so that only one trajectory is decoded.

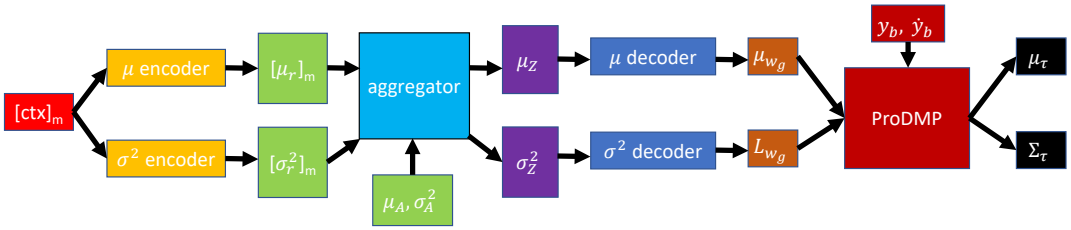


Figure 4.20: Architecture with PB decoder.

The second, Monte-Carlo (MC) architecture, was based on taking the action sample directly in the latent space instead of taking the sample from the space of ProDMP weights and goal. This architecture is presented in Figure 4.21. It also takes posterior distribution as the starting point for the decoding process. From this distribution it takes n samples $[\boldsymbol{z}]_n$ and they are input to one decoder network that has weights and goal as the output: $[\boldsymbol{w}_g]_n$. Variance of the posterior is directly forwarded to the second decoder that outputs the distribution of the weights and goal - $\boldsymbol{\sigma}^2_{w_g}$. From these two values, ProDMPs are able to calculate m trajectories $[\boldsymbol{\tau}]_m$ and variance $\boldsymbol{\sigma}^2_\tau$. Cost function used to train this network was negative marginal log-likelihood. This function increases probability of observing ground-truth trajectory given the several (n) samples from the distribution and variance (both determined by decoder parameters $\boldsymbol{\theta}$):

$$\mathcal{C}(\boldsymbol{\tau}, \boldsymbol{y}_b, \dot{\boldsymbol{y}}_b, \mathbf{ctx}) = -\log \mathcal{N}(\boldsymbol{y}|[\boldsymbol{\tau}]_{\boldsymbol{n}}, \boldsymbol{\sigma}^{\mathbf{2}}_{\boldsymbol{\tau}}, \boldsymbol{\theta}). \tag{4.21}$$

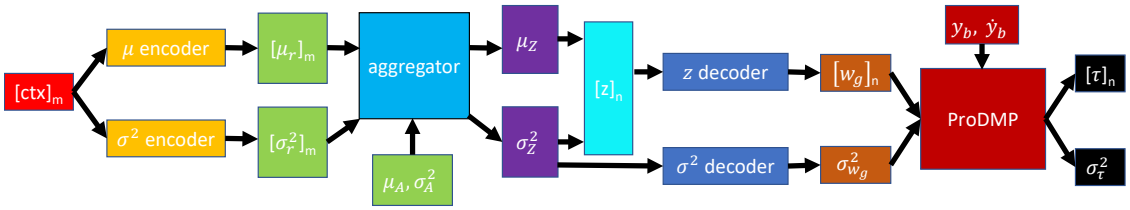


Figure 4.21: Architecture with MC decoder.

An important thing to note with this architecture is that the trajectory is sampled in the latent space distribution. Consequently, when using the forward pass in the evaluation, only z decoder is used and only one trajectory is predicted.

The last part of our architecture is based on ProDMPs. This new method [56] enables for combining the benefits from both Dynamic Movement Primitives (DMPs) [47] and Probabilistic Movement Primitives (ProMPs) [55]. Having DMP weights and goals determined by decoder network and given the initial position and velocity as the initial conditions $(\boldsymbol{y}_b, \dot{\boldsymbol{y}}_b)$, this method is able to provide trajectory distribution given as a multivariate Gaussian distribution.

4.3.2 Experimental protocol and results

In the first step of our research, the complete database of the human trajectories was collected. For this purpose, we used a simulated Panda Franka robot in MuJoCo environment. Using the computer mouse for kinesthetic guidance of the robot, we acquired the database of 300 trajectories and stored all the relevant data - context points. All the trajectories lasted 2 seconds.

This database was further used to train both architectures presented in Figures 4.20 and 4.21. Both mean and variance encoder consisted of the input layer size of 8 that corresponds to the dimensionality of our context. All the encoders used consisted of 128 neurons in 3 hidden layers. Output size, determining also the latent space size, was chosen to be $S \in \{8, 16\}$. Consequently, decoders had input size S. Three hidden layers consisting of 128 neurons were used here just as with the encoders, and output size was set to be 52. This was caused by us choosing for both degrees of freedom to use 25 radial basis functions in ProDMP and one goal position. Activation function used in our case was Leaky-ReLu.

We trained both architectures (PB+agg and MC+agg) with and without aggregation (PB and MC). All the training was done in 10000 epochs and (for the sake of completeness of the results) in 10 different seeds. For the cases where aggregation was used, we used 5 last context points, while in other cases we used only the last. For both cases of using MC decoders, we used $n = 16$ Monte-Carlo samples for choosing the latent space variable. For each randomly chosen point of the trajectory, only the following 0.5 s were considered when training. When evaluating the trained architectures, we considered the following cases:

- **MC + agg (mean)**: MC architecture combined with aggregation was used with 5 context points in both training and evaluation. In the evaluation case, only mean z variable was used and decoded.

- **MC (mean)**: MC architecture without aggregation was used in both training and evaluation. In the evaluation case, only mean z variable was used and decoded.

- **PB + agg (mean)**: PB architecture combined with aggregation was used with 5 context points in both training and evaluation. In the evaluation case, only mean of the trajectory distribution was used.

- **PB (mean)**: PB architecture without aggregation was used in both training and evaluation. In the evaluation case, only mean of the trajectory distribution was used.

- **MC + agg (stochastic)**: MC architecture combined with aggregation was used with 5 context points in both training and evaluation. In the evaluation case, z variable was sampled from the distribution and decoded.

- **MC (stochastic)**: MC architecture without aggregation was used in both training and evaluation. In the evaluation case, z variable was sampled from the distribution and decoded.

- **PB + agg (stochastic)**: PB architecture combined with aggregation was used with 5 context points in both training and evaluation. In the evaluation case, trajectory was sampled from the final trajectory distribution.

- **PB (stochastic)**: PB architecture without aggregation was used in both training and evaluation. In the evaluation case, trajectory was sampled from the final trajectory distribution.

In the evaluation of the learned policy, neural network was always predicting the following 0.5 s of the trajectory based on the provided context. Total allowed length of the trajectory was 4 s, thus leading to 8 replanning steps.

Figure 4.22 shows the average error in the box position and orientation for the latent space sizes of 8 and 16. Higher latent space sizes were also tested, but they did not bring any increase in the performance. Lower latent space sizes were excluded because they were insufficient to catch the complexity of the observation. From the graphs, it can be seen that using the mean rather than randomly sampled variable resulted in better performance. Also, it is noticeable that aggregation increases the performance when using the PB architecture while the difference when applied on MC architecture was much lower. The best performance was observed when using the MC+agg and PB+agg methods.

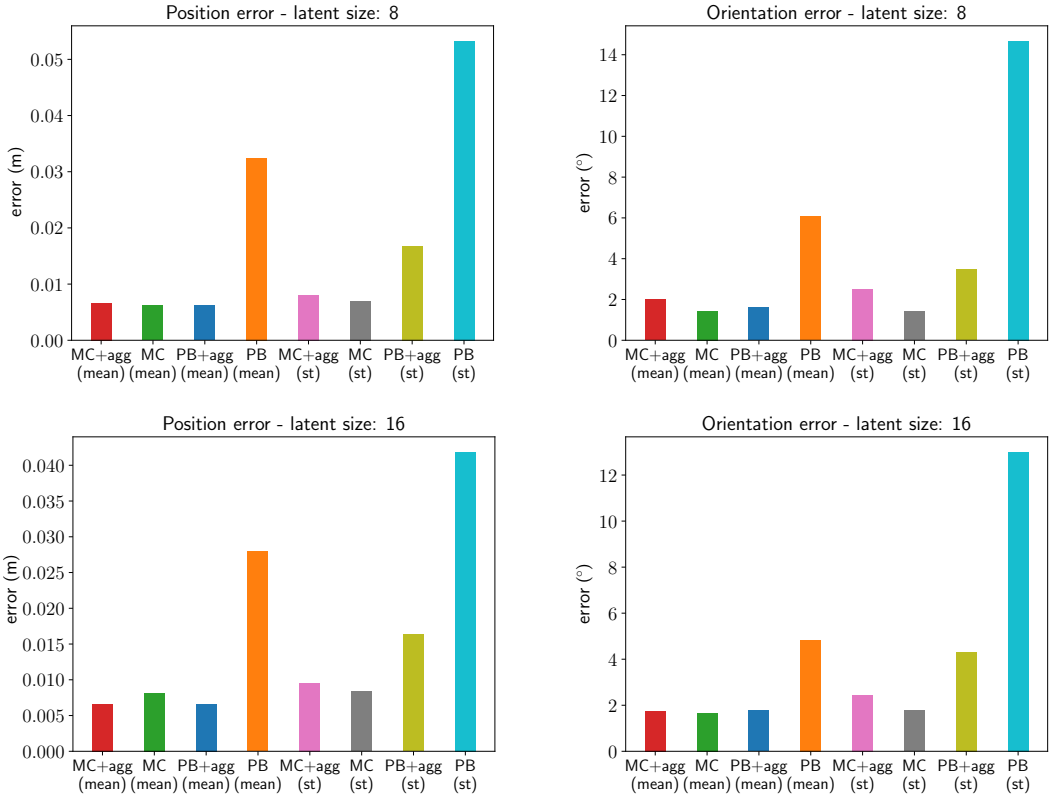


Figure 4.22: Bar graphs showing the average error in position and orientation for all the tested cases for latent space sizes 8 and 16.

The performance was tested for all the experiments in 10 different seeds and with using 20 test trajectories. Histograms showing the distribution of the error in position and orientation are shown in Figure 4.23.

Since stochastic methods are, due to their diversity in the data, important for the transferability of the learned policy to the situations where the parameters of the task are slightly different (changed friction or mass of the box), we additionally tested PB+agg method and MC+agg method for the error increase with respect to the increase in standard deviation of the trajectories.

Average error is given as $\sum_{s=1}^{S}\sum_{k=1}^{K}(|e|_{s,k}^{x} + |e|_{s,k}^{y} + 0.1|e|_{s,k}^{\varphi})$, where S is the number of seeds, K is the number of test trajectories for each seed and $|e|_{s,k}^{x}$, $|e|_{s,k}^{y}$, $|e|_{s,k}^{\varphi}$ are errors in the final box position in x, y axis given in meters and orientation about z axis

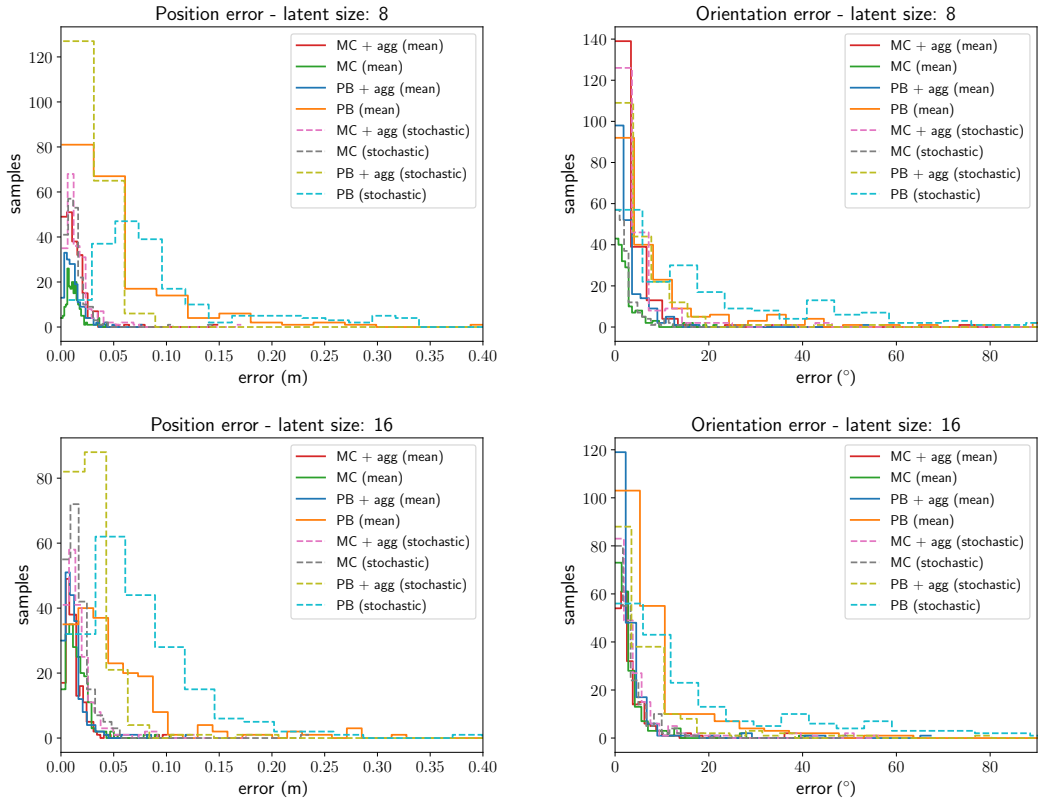


Figure 4.23: Histogram showing the distribution of the error in position and orientation for different latent space size.

given in degrees, respectively. For obtaining error increase as a consequence of standard deviation, entropy of the MC policy was varied by multiplying the standard deviation σ_z^2 with coefficients $[0, 0.1, 0.5, 1.0, 2.0, 5.0, 10, 15]$ and for the case of PB multiplying Cholesky of the covariance matrix with coefficients $[0, 0.1, 0.5, 1.0, 2.0, 5.0]$. Larger coefficients consequently resulted in higher standard deviation of the resulting trajectories. Graphs in Figure 4.24 show that MC+agg method offers solutions of higher diversity while maintaining the smaller error. This makes it more suitable for acquiring the policy that will be later transferred into the slightly different environment [57].

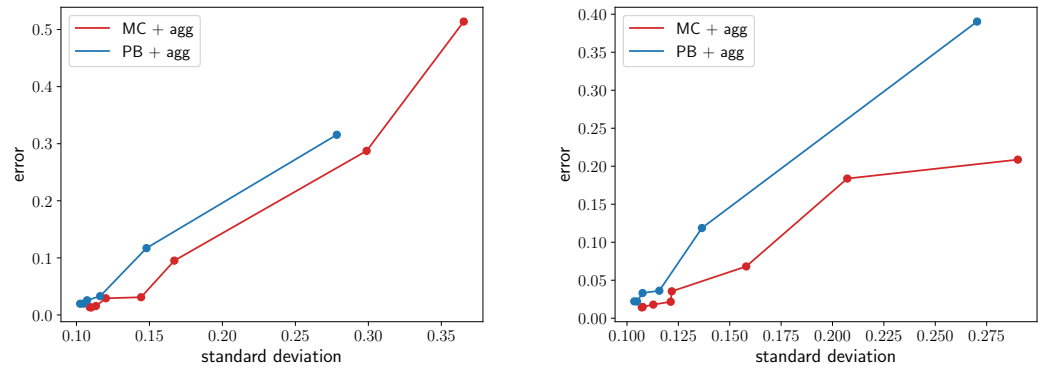


Figure 4.24: Error with respect to the increase in standard deviation of the learned policy.

4.4 Discussion

In this chapter, we have shown and evaluated methods that enable faster and safer learning of a skill on a robot. Learning from scratch in the source domain is not in the scope of this chapter, but the main focus is on the application of the previously obtained knowledge to the target environment. We have presented two different deterministic approaches that managed to successfully adapt the knowledge in the reasonable and real-world realistic number of iterations. Furthermore, we have shown how we can store the knowledge and how we can accelerate the adaptation of this knowledge using RL instead of the traditional backpropagation. This is the most complicated case in this thesis and therefore it is put on the bottom of our pyramid. With having a successful policy implemented on the real robot, each of the separate trajectories can be further optimized (filtered, refined to satisfy minimal-jerk criteria or made shorter) using RL or approaches similar to the approach from Chapter 2. Also, they can provide us with planning of the specific trajectories for a subset tasks and replace planners from Chapter 3.

The topic of discussion is generality of the proposed approaches, which was applied only for a single task and with a lot of user-knowledge input into the preparation. An example of input user knowledge is the even spread of the targets for learning. The overall approach is not different from transfer learning for visual tasks, where one utilizes the already learned feature extraction of a given, trained network [34]. Similar to this approach, they take a pretrained network and use it as a starting point to learn a new task much faster. This new task might be something very specific, for example classification of some new, previously not learned objects. Even in this case the classification will be better if the images of the new objects cover as much viewpoints and diversity as possible. The latter are again provided by the user.

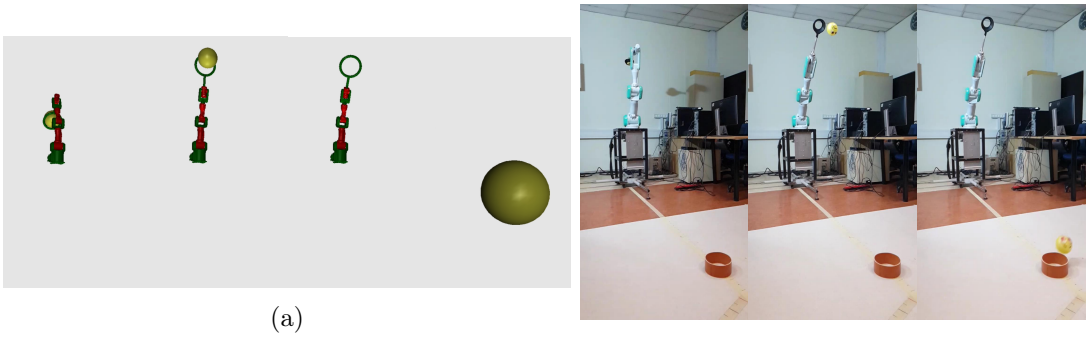
However, there is still a difference in robotic movement generation and images: For the robotic actions, there are RL methods of learning and obtaining the dataset from scratch. Therefore, we introduced probabilistic methods that (due to their higher resistance to changed environment) provide a baseline for the learning where less user knowledge will be needed when performing TL.

Throwing with robot arm

The presented approach has shown that even a smaller number of required executions in the target domain (28% smaller for the case of 16 targets and 46% for 12 targets) was enough to outperform conventional backpropagation approach (Figure 4.7). The approach can be further extended, by adding another layer, and having three or even more consecutive domains: kinematic $\mapsto$ dynamic $\mapsto$ real-world.

The proposed approach provides a method for sim-to-real transfer learning with domain adaptation, where RL is applied to modify the most changed neural network parameters. Thus, the changing of these parameters is affected by both retraining and RL, which acts similar as freezing of a part of the network, because the most changed part is changed much more. Application to robot throwing and transfer from simple kinematic to complex dynamic simulation, which stands in place of the real-world, has shown that the approach can considerably accelerate the learning in the adapted domain.

In the future we will test the approach in consecutive kinematic $\mapsto$ dynamic $\mapsto$ real-world mapping. The possibility of executing initial policy in the real-world has shown great potential and TL from the dynamic simulation (Figure 4.25a) to the real-world (Figure 4.25b) should behave similar as in the presented sim-to-sim environment. Despite dealing with only several hundred target environment throws, the task has turned out to be quite challenging.



(a) (b)

Figure 4.25: Experimental setup for evaluation of TL. (a) Mujoco dynamic simulation. (b) Mitsubishi PA-10 robot in its initial posture (left), after the release of the ball (center), and when the ball lands (right).

Throwing with humanoid robot

The proposed few-shot transfer learning in this section was shown to be faster than simply applying backpropagation over the complete NN skill model, and faster than using RL or BP on one layer of the NN model, while keeping the other layers intact. The results show that roughly 20 to 30 iterations were needed to reach a learning plateau. Since 25 throws were applied in one iteration, that still amounts to a lot of throws. But, learning in the source domain required roughly 3000 throws, thus the result of transfer learning requires an order of magnitude less throws.

The RL+BP experiment showed the best results, but they were hardly any better than BP-r. Here it should be noted that the structure of the DNN can have a considerable effect on the applicability of this approach. For example, an encoder-decoder type of NN, where RL was applied to the smallest, latent-space layer, could probably have a much larger effect. However, such a network would require a larger source domain database to train. Testing different NN skill model network structures remains part of our future work.

Real-world implementation, shown in Figure 4.26, clearly shows the capability of TALOS robot throwing the ball in the same configuration as in simulation. Given the complexity of the TALOS robot, not enough were implemented to show actual sim-to-real transfer.

The proposed few-shot sim-to-real transfer learning using RL+BP approach on only one layer of the end-to-end DNN skill model has shown to accelerate learning in the target domain, and has outperformed other tested methods. In the future, we will test different DNN structures and fully evaluate the approach on the real robot.

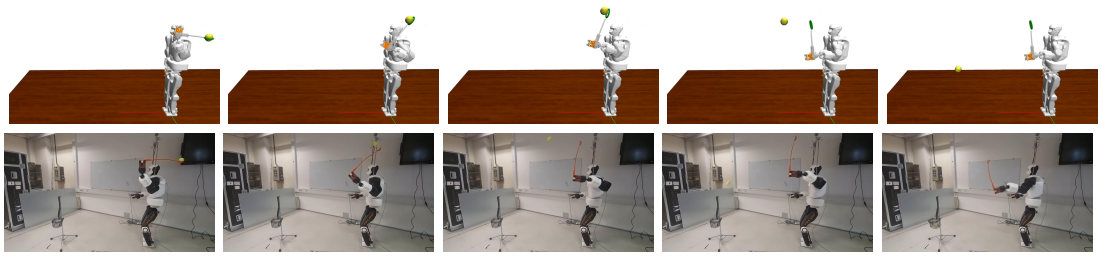


Figure 4.26: Image sequence of a throw with TALOS humanoid robot in MuJoCo simulation environment, and in the real-world. Both represent the target domain, but only initial throwing action for transfer learning to the real-world has been obtained.

Probabilistic approaches

Presented probabilistic approaches show the potential benefits of pairing the ProDMP approach with neural network in order to obtain successful imitation learning policies for dynamic tasks. Since, once trained, these methods can provide the whole spectre of different solutions, they are more probable to find the right solution quicker and therefore converge faster in the adaptation phase to the new domain. This is due to their exploration phase not being based on random sampling only, but also included in the probabilistic nature of the predicted trajectories. Moreover, since they are also trained on the same task in the different environment, they are not that much dependent on setting up the exploration noise, and therefore it is less probable that some parameters will produce the trajectory that is not executable. Additionally, we have shown benefits of sampling the trajectory directly in the latent space of encoder-decoder network and positive influence of aggregation to the final policy. We plan to continue research in the direction of using probabilistic approaches for few-shot transfer learning to a new environment. In the future, we intend to use the MC or MC+agg models since they will provide higher precision than PB or PB+agg models, as shown in results, while also having higher diversity of the data, as shown in Figure 4.24, thus making the TL easier as reported in [57].

Chapter 5

Discussion and Outlook

This thesis focuses on the exploration and development of techniques for learning autonomous robot behaviours. Throughout the presented research, we have utilized a top-down approach through our learning-division pyramid, which allowed us to introduce various levels of robot autonomy. As we progressed from the top to the bottom of the pyramid, the complexity of autonomous behaviour generation was increasing. For each level, we provided illustrative examples to demonstrate the capabilities and applications of autonomous robots. Additionally, we highlighted notable advancements in the field corresponding to each level of autonomy.

First, we started from the top of our pyramid focusing on the most straightforward task where the trajectory was known but not optimal. We introduced an automatized method for extracting and parametrizing the trajectories from a CAD model of an object and used CDMPs with explicitly defined velocity profile. We further applied a model-free learning approach that solely relies on the sharpness of the image as a learning criterion. Our method offers the advantage of expediting the setup process of traditional factory lines used for large-scale serial production. It assumes that the environmental conditions remain constant and unchanged throughout the production lifespan. It is worth noting that even in this task, achieving complete autonomy was not possible due to the requirement for the user to manually select the path and breakpoints on the CAD model. In our future work, we intend to address these challenges. While the user will still need to choose the path, we aim to automate the process of segmenting it into twice differentiable segments. Furthermore, as the availability of CAD models is not always guaranteed (particularly in older production lines or cases where it is considered proprietary information), we plan to employ neural networks capable of image processing to obtain the model and initial trajectory.

Moving on to the middle segment of the learning-division pyramid, we delved into the complexities of accelerating an entire industrial process. As an example, quality inspection with an in-hand camera task was chosen. This case presented a higher level of complexity compared to the previous level, since the trajectories were unknown. However, we possessed substantial knowledge about the task at hand. Specifically, we were aware of all the desired goal positions, which were finite in number. Additionally, we had access to a model of the robot and its environment and a manually labelled dataset consisting of defective parts and their corresponding production line parameters. We used the latent space of the AE network to cluster different types of errors. Based on the knowledge of the camera-object relationship, we were able to use the motion planners to generate trajectories. It is important to note that the generated trajectories were not optimal in terms of time or path distance. However, these trajectories could be iteratively optimized using RL or a similar optimization process as used at the top of the pyramid. Despite the existence of numer-

ous potential trajectories, they are frequently executed throughout the production cycle. As the setup remains unchanged throughout the entire production lifespan, a database of transition trajectories can be constructed. RL can use past experience stored in this database to iteratively update the trajectories, resulting in a more optimized production line over time. In the future, we plan to implement an optimization method that incorporates the collected database. Additionally, we aim to explore the possibility of training NN to execute motions in a more optimal manner compared to traditional planners. This step would come after the completion of RL optimization. The intention is to replace the storage-intensive database utilized during the optimization process with a trained NN. This replacement would not only reduce the storage requirements but also enable faster planning times, as the forward pass through the NN is generally quicker than traditional planners. By leveraging the advantages of NNs, we aim to enhance the efficiency and effectiveness of motion planning in the context of autonomous robot behaviour.

In the bottom part of our learning-divided pyramid, our efforts were directed towards advancing approaches for achieving fully autonomous learning. However, it is important to note that this thesis did not specifically cover the learning of complete skills from scratch in simulation of a robot, as this topic has already been extensively researched and has yielded remarkable results. Instead, our primary focus was on advancing transfer learning (TL) methodologies to facilitate the transition of knowledge from simulated environments to real-world applications. We introduced two deterministic methodologies for storing knowledge acquired from source environments and conducted a benchmark study to evaluate the effectiveness of combining RL and backpropagation techniques in efficiently transferring this knowledge to the target environments. By evaluating different TL methodologies, we aimed to bridge the gap between simulated learning and real-world applications, thus taking a significant step towards practical implementation of autonomous learning in real-world scenarios. It is important to acknowledge that these approaches still rely on human knowledge to be provided as prior information during the learning process. In our case, we needed to have an understanding of the realistic throwing range of the robot and ensure that the targets were evenly distributed in space. This prior knowledge was essential for defining the reward function, which aimed to optimize results across the entire range of tasks rather than overfitting to specific instances. By incorporating human knowledge into the learning process, we aimed to guide and shape the learning algorithms to achieve faster learning and to ensure that the learned behaviours were optimized in a meaningful and practical manner. In our future work, we aim to remove the reliance on prior information and make the learning process more autonomous. One of the approaches we plan to explore is presented in the Section 4.3 and it is based on AE networks combined with probabilistic representations of trajectories [55], [56]. This approach aims to enhance the flexibility and adaptability of the learning process. Once trained, these methods are more resistant to changes in the environment and require less tuning of RL algorithm parameters such as exploration noise. This resistance is caused by the fact that for the small changes in the environment, their distribution already contains the required solution. Therefore, TL is based only on changing the mean until all the predicted actions from the distribution are capable of successfully performing the task.

We are currently working on developing an architecture that will provide a pipeline for the full end-to-end learning with the capabilities of TL without adding the additional human knowledge into the adaptation [63]. The focus of our future work lies in applying transfer learning (TL) techniques to various tasks to demonstrate that the learned knowledge can be effectively transferred and generalized. By pursuing these research directions, we aim to advance the autonomy of robot learning and promote the ability to acquire behaviours without heavy reliance on predefined information or specific prior works.

Chapter 6

General Summary and Conclusions

All the initially specified hypotheses and goals were successfully addressed in the thesis. Moreover, we provided a classification of the autonomous behaviour based on the complexity as well as the frequency of their usage.

In the first chapter, we have successfully developed an automatized method for extracting the path and parametrizing the trajectory in order to encode it in the parametric form of CDMPs that can be further optimized with respect to time, as stated in our first contribution. We also introduced the combination of ILC and RWPL that is capable of successfully tuning the robot velocity by taking image sharpness as an input better than other, more widely used learning algorithms. This way, we have confirmed our hypothesis that autonomous learning such as reinforcement learning and iterative learning control can be used for the optimization and that the motion acquired this way is faster and easier to acquire than human-calculated motion.

In the second chapter, we have shown that motion planners are capable of generating the trajectories online and that they are not a bottleneck in the adaptive industrial processes as stated in our second hypothesis. Since the general solution for choosing the right motion planner does not exist and not all the planners are fast enough, we have presented a comprehensive method of benchmarking and choosing the right one. In the same chapter, we have demonstrated that AI is capable of predicting the possible defects based on the process parameters. We analyzed the data from the functional production line and exploited AI methods to determine the points from which the part needs to be checked. Precisely, we employed AE network to reduce the dimensionality and unsupervised learning to cluster the parts in its latent space. With successful clustering, we have shown that we can approximate which error is the most likely to happen. A combination of AE error prediction and fast motion planners has confirmed our second hypothesis that the industrial processes can be adapted and accelerated autonomously.

The third part of the thesis focuses on methods for closing the sim-to-real gap. Although not the main topic, we have presented the methods that helped us acquire the knowledge in the source environment. By using different neural network structures, we have managed to store this knowledge from the source environment. Despite not performing great, this knowledge was initially executable in the target environment (both in the dynamic simulation and on the real-robot) and therefore we prove that this knowledge is not wasted, but it can be reused to jump-start learning in the changed environment as stated in our fourth hypothesis. In the same chapter, we developed two architectures where RL successfully accelerated the adaptation process compared to classical backpropagation method as stated in our fifth hypothesis. With this, we managed to perform adaptation in the reasonable number of target environment executions and therefore achieved a step further towards fully autonomous learning. Additionally, we demonstrated and benchmarked

the quality of the source environment policy for two different probabilistic approaches. Since it is proven in [57] that these methods are more robust to the environment change, and they also remove strict dependency of the manual tuning of the hyperparameters used in algorithms, we plan to examine the possibilities of using them for advancing the methods for closing the sim-to-real gap in the more unified and general manner.

References

[1] T. Gašpar, B. Ridge, R. Bevec, *et al.*, "Rapid hardware and software reconfiguration in a robotic workcell," in *IEEE 18th International Conference on Advanced Robotics (ICAR)*, 2017, pp. 229–236.

[2] T. Gašpar, M. Deniša, P. Radanovič, *et al.*, "Smart hardware integration with advanced robot programming technologies for efficient reconfiguration of robot workcells," *Robotics and Computer-Integrated Manufacturing (RCIM)*, vol. 66, no. 101979, 2020.

[3] Y. Koren, "Reconfigurable manufacturing system," in *CIRP Encyclopedia of Production Engineering*, S. Chatti, L. Laperrière, G. Reinhart, and T. Tolio, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 1417–1423.

[4] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007.

[5] J. Silvério and S. Calinon, "Robot learning in factories of the future: Adaptability, redundancy and exploitation of movement options," in *IROS Workshop: Factory of the Future: How to digitalize the robot-aided manufacturing process in Industry 4.0?*, Macau, China, 2019.

[6] J. Kim and E. Croft, "Online near time-optimal trajectory planning for industrial robots," *Robotics and Computer-Integrated Manufacturing (RCIM)*, vol. 58, pp. 158–171, 2019.

[7] D. Meike, M. Pellicciari, and G. Berselli, "Energy efficient use of multirobot production lines in the automotive industry: Detailed system modeling and optimization," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 3, pp. 798–809, 2014.

[8] L. Žlajpah, "On time optimal path control of manipulators with bounded joint velocities and torques," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, Minneapolis, MN, USA, 1996, 1572–1577 vol.2.

[9] Z. Miljković and M. Petrović, "Application of modified multi-objective particle swarm optimisation algorithm for flexible process planning problem," *International Journal of Computer Integrated Manufacturing*, vol. 30, no. 2-3, pp. 271–291, 2017.

[10] L. Pérez, Í. Rodríguez, N. Rodríguez, R. Usamentiaga, and D. F. García, "Robot guidance using machine vision techniques in industrial environments: A comparative review," *Sensors (Basel, Switzerland)*, vol. 16, no. 335, 2016.

[11] H. Wang, J. Wang, W. Chen, and L. Xu, "Automatic illumination planning for robot vision inspection system," *Neurocomputing*, vol. 275, pp. 19–28, 2018.

[12] N. Herakovič, "Robot vision in industrial assembly and quality control processes," in *Robot Vision*, Rijeka: IntechOpen, 2010, ch. 26, pp. 501–534.

[13] Che-Seung Cho, Byeong-Mook Chung, and Moo-Jin Park, "Development of real-time vision-based fabric inspection system," *IEEE Transactions on Industrial Electronics*, vol. 52, no. 4, pp. 1073–1079, Aug. 2005.

[14] D. Schreiber, L. Cambrini, J. Biber, and B. Sardy, "Online visual quality inspection for weld seams," *The International Journal of Advanced Manufacturing Technology*, vol. 42, no. 5, pp. 497–504, 2009.

[15] R. Bevec, T. Gašpar, and A. Ude, "Robot-driven autofocus control mechanism for an in-hand fixed focus camera," in *Advances in Service and Industrial Robotics*, N. A. Aspragathos, P. N. Koustoumpardis, and V. C. Moulianitis, Eds., Cham: Springer International Publishing, 2019, pp. 551–559.

[16] Z.-G. Wu, C.-Y. Lin, H.-W. Chang, and P. T. Lin, "Inline inspection with an industrial robot (IIIR) for mass-customization production line," *Sensors*, vol. 20, no. 11, 2020.

[17] M. Montironi, P. Castellini, L. Stroppa, and N. Paone, "Adaptive autonomous positioning of a robot vision system: Application to quality control on production lines," *Robotics and Computer-Integrated Manufacturing (RCIM)*, vol. 30, no. 5, pp. 489–498, 2014.

[18] Z. Lončarević, S. Reberšek, A. Ude, and A. Gams, "Randomized robotic visual quality inspection with in-hand camera," in *Intelligent Autonomous Systems (IAS) 17*, I. Petrovic, E. Menegatti, and I. Marković, Eds., Cham: Springer Nature Switzerland, 2023, pp. 483–494.

[19] T. Mac, C. Copot, D. Tran, and R. Keyser, "Heuristic approaches in robot path planning: A survey," *Robotics and Autonomous Systems (RAS)*, vol. 86, pp. 13–28, 2016.

[20] A. Khan, C. Mineo, G. Dobie, C. Macleod, and S. Pierce, "Vision guided robotic inspection for parts in manufacturing and remanufacturing industry," *Journal of Remanufacturing*, vol. 11, Apr. 2021.

[21] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Path planning and trajectory planning algorithms: A general overview," in *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*, G. Carbone and F. Gomez-Bravo, Eds. Cham: Springer International Publishing, 2015, pp. 3–27.

[22] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2013.

[23] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[24] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3826–3839, 2020.

[25] R. Pahič, A. Gams, and A. Ude, "Reconstructing spatial aspects of motion by image-to-path deep neural networks," *IEEE Robotics and Automation Letters (RAL)*, vol. 6, no. 1, pp. 255–262, 2021.

[26] Z. Lončarević, R. Pahič, A. Ude, and A. Gams, "Generalization-based acquisition of training data for motor primitive learning by neural networks," *Applied Sciences*, vol. 11, no. 3, p. 1013, 2021.

[27] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, and J. Peters, "Robot learning from randomized simulations: A review," *Frontiers in Robotics and AI*, vol. 9, 2022.

[28] Y. Chebotar, A. Handa, V. Makoviychuk, *et al.*, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8973–8979.

[29] A. Gams, S. A. Mason, A. Ude, S. Schaal, and L. Righetti, "Learning task-specific dynamics to improve whole-body control," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2018, pp. 280–283.

[30] E. Salvato, G. Fenu, E. Medvet, and F. A. Pellegrino, "Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning," *IEEE Access*, vol. 9, pp. 153 171–153 187, 2021.

[31] Z. Lončarević, A. Ude, and A. Gams, "Accelerated robot skill acquisition by reinforcement learning-aided sim-to-real domain adaptation," in *20th International Conference on Advanced Robotics (ICAR)*, 2021, pp. 269–274.

[32] R. Pahič, Z. Lončarević, A. Ude, B. Nemec, and A. Gams, "User feedback in latent space robotic skill learning," in *IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, Nov. 2018, pp. 270–276.

[33] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1345–1359, 2010.

[34] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744.

[35] S. James, P. Wohlhart, M. Kalakrishnan, *et al.*, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *Conference on Computer Vision and Pattern Recognition*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2019, pp. 12 619–12 629.

[36] K. Bousmalis, A. Irpan, P. Wohlhart, *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 4243–4250.

[37] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Machine learning*, vol. 84, no. 1–2, pp. 171–203, Jul. 2011.

[38] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[39] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[40] H. van Hoof, N. Chen, M. Karl, P. van der Smagt, and J. Peters, "Stable reinforcement learning with autoencoders for tactile and visual data," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Deajeon, Korea, 2016, pp. 3928–3934.

[41] N. Chen, J. Bayer, S. Urban, and P. van der Smagt, "Efficient movement representation by embedding dynamic movement primitives in deep autoencoders," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Seoul, Korea, 2015, pp. 434–440.

[42] L. Le, A. Patterson, and M. White, "Supervised autoencoders: Improving generalization performance with unsupervised regularizers," in *Advances in Neural Information Processing Systems 31*, Curran Associates, 2018, pp. 107–117.

[43] N. Chen, M. Karl, and P. V. D. Smagt, "Dynamic movement primitives in latent space of time-dependent variational autoencoders," *IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, vol. 2, no. 3, pp. 629–636, 2016.

[44] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[45] Z. Lončarević, "Robotic learning in feature space of deep autoencoders," M.S. thesis, Jožef Stefan International Postgraduate School, 2019.

[46] Z. Lončarević, M. Simonič, A. Ude, and A. Gams, "Combining reinforcement learning and lazy learning for faster few-shot transfer learning," in *IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, 2022, pp. 285–290.

[47] A. Ijspeert, J. Nakanishi, P. Pastor, H. Hoffmann, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.

[48] A. Ude, B. Nemec, T. Petrič, and J. Morimoto, "Orientation in cartesian space dynamic movement primitives," in *International Conference on Robotics and Automation (ICRA)*, 2014, pp. 2997–3004.

[49] T. Gašpar, B. Nemec, J. Morimoto, and A. Ude, "Skill learning and action recognition by arc-length dynamic movement primitives," *Robotics and Autonomous Systems (RAS)*, vol. 100, pp. 225–235, 2018.

[50] K. L. Moore, "Iterative learning control: An expository overview," in *Applied and Computational Control, Signals, and Circuits: Volume 1*, B. N. Datta, Ed. Boston, MA: Birkhäuser Boston, 1999, pp. 151–214.

[51] D. Bristow, M. Tharayil, and A. Alleyne, "A survey of iterative learning," *Control Systems*, vol. 26, pp. 96–114, 2006.

[52] R. Vuga, B. Nemec, and A. Ude, "Speed adaptation for self-improvement of skills learned from user demonstrations," *Robotica*, vol. 34, no. 12, pp. 2806–2822, 2016.

[53] O. Sigaud and F. Stulp, "Policy search in continuous action domains: An overview," *Neural Networks*, vol. 113, pp. 28–40, 2019.

[54] Z. Lončarević, A. Gams, S. Reberšek, *et al.*, "Specifying and optimizing robotic motion for visual quality inspection," *Robotics and Computer-Integrated Manufacturing (RCIM)*, vol. 72, p. 102 200, 2021.

[55] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., vol. 26, Curran Associates, Inc., 2013.

[56] G. Li, Z. Jin, M. Volpp, F. Otto, R. Lioutikov, and G. Neumann, *ProDMPs: A unified perspective on dynamic and probabilistic movement primitives*, 2022. arXiv: 2210.01531 [cs.RO].

[57] B. Eysenbach and S. Levine, *Maximum entropy RL (provably) solves some robust RL problems*, 2021. DOI: 10.48550/ARXIV.2103.06257.

[58] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics and Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec. 2012.

[59] M. Moll, I. A. Sucan, and L. E. Kavraki, "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization," *IEEE Robotics and Automation Magazine*, vol. 22, no. 3, pp. 96–102, 2015.

[60] H. Mir, P. Xu, and P. van Beek, "An extensive empirical evaluation of focus measures for digital photography," in *Digital Photography X*, International Society for Optics and Photonics, vol. 9023, Spie, 2014, pp. 167–177.

[61] R. Pahič, Z. Lončarević, A. Gams, and A. Ude, "Robot skill learning in latent space of a deep autoencoder neural network," *Robotics and Autonomous Systems (RAS)*, vol. 135, p. 103 690, 2021.

[62] Z. Lončarević, T. Petrič, and A. Gams, "Fitting constrained trajectory with high variability into redundant robot workspace," in *Advances in Service and Industrial Robotics*, A. Müller and M. Brandstötter, Eds., Springer, Cham: Springer International Publishing, 2022, pp. 167–175.

[63] Z. Lončarević, G. Li, G. Neumann, and A. Gams, "An encoder-decoder architecture for smooth motion generation," in *Advances in Service and Industrial Robotics*, T. Petrič, A. Ude, and L. Žlajpah, Eds., Cham: Springer Nature Switzerland, 2023, pp. 358–366.

[64] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. Hoboken, NJ: John Wiley & Sons, 2006.

[65] *Creo parametric 7.0*, 121 Seaport Blvd Boston, MA 02210, USA: PTC Inc, 2020.

[66] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, 2010.

[67] A. Gams, S. Reberšek, B. Nemec, *et al.*, "Robotic learning for increased productivity: Autonomously improving speed of robotic visual quality inspection," in *IEEE International Conference on Automation Science and Engineering (CASE)*, Vancouver, BC, Canada, 2019, pp. 1275–1281.

[68] Z. Lončarević, R. Pahič, and A. Gams, "Learning of the velocity profile for quality inspection motion using PoWER," in *29th International Electrotechnical and Computer Science Conference*, 2020, pp. 156–159.

[69] P. Kormushev, S. Calinon, R. Saegusa, and G. Metta, "Learning the skill of archery by a humanoid robot iCub," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2010, pp. 417–423.

[70] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: A Bradford Book, 2018.

[71] T. T. Andersen, "Optimizing the Universal Robots ROS driver," Technical University of Denmark, Department of Electrical Engineering, Tech. Rep., 2015.

[72] H.-Y. Lin, K.-J. Li, and C.-H. Chang, "Vehicle speed detection from a single motion blurred image," *Image and Vision Computing*, vol. 26, pp. 1327–1337, Oct. 2008.

[73] J. Villalba-Diez, D. Schmidt, R. Gevers, J. Ordieres-Meré, M. Buchwitz, and W. Wellbrock, "Deep learning for industrial computer vision quality control in the printing Industry 4.0," *Sensors (Basel, Switzerland)*, vol. 19, no. 18, 2019.

[74] V. Zavrtanik, M. Kristan, and D. Skočaj, "Reconstruction by inpainting for visual anomaly detection," *Pattern Recognition*, vol. 112, p. 107 706, 2021.

[75] H. Choset, K. Lynch, S. Hutchinson, *et al.*, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, May 2005.

[76] A. Abbadi and R. Matousek, "Path planning implementation using Matlab," in *International Conference of Technical Computing*, Bratislava, Slovakia, Nov. 2014.

[77] B. Patle, G. L, D. A. Pandey, D. Parhi, and J. Anne, "A review: On path planning strategies for navigation of mobile robot," *Defence Technology*, 2019.

[78] Mehmet Akif AKKUŞ, *Cell decomposition*, `https://user.ceng.metu.edu.tr/~akifakkus/courses/ceng786/trapezoidalDecomposition.html`, Last accessed on 1.6.2023.

[79] F. Raheem and M. M.Badr, "Development of modified path planning algorithm using artificial potential field (APF) based on PSO for factors optimization," *American Scientific Research Journal for Engineering, Technology, and Sciences*, vol. 37, pp. 316–328, 2017.

[80] L. Kavraki and J.-. Latombe, "Randomized preprocessing of configuration space for path planning: Articulated robots," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 1994, pp. 1764–1771.

[81] J. Lee, O. Kwon, L. Zhang, and S. Yoon, "A selective retraction-based RRT planner for various environments," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 1002–1011, 2014.

[82] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[83] A.-M. Zou, Z.-G. Hou, S.-Y. Fu, and M. Tan, "Neural networks for mobile robot navigation: A survey," in *Advances in Neural Networks*, 2006, pp. 1218–1226.

[84] H.-H. Kim, Y.-S. Ha, and G.-G. Jin, "Environmental map building for a mobile robot using infrared range-finder sensors," in *Advanced Robotics*, Nov. 2003, pp. 711–716.

[85] H. M. Barbera, A. G. Skarmeta, M. Z. Izquierdo, and J. B. Blaya, "Neural networks for sonar and infrared sensors fusion," in *Proceedings of the Third International Conference on Information Fusion*, vol. 2, 2000, WEB4/18–WEB4/25 vol.2.

[86] N. Vlassis, Y. Motomura, and B. Krose, "Supervised dimension reduction of intrinsically low-dimensional data," *Neural computation*, pp. 191–215, 2002.

[87] S. Dezfoulian, D. Wu, and I. Ahmad, "A generalized neural network approach to mobile robot navigation and obstacle avoidance," *Advances in Intelligent Systems and Computing*, pp. 25–42, 2013.

[88] Z. Xu, X. Zhou, and S. Li, "Deep recurrent neural networks based obstacle avoidance control for redundant manipulators," *Frontiers in Neurorobotics*, p. 47, 2019.

[89] D. Janglová, "Neural networks in mobile robot motion," *International Journal of Advanced Robotic Systems*, 2004.

[90] C. Kozakiewicz and M. Ejiri, "Neural network approach to path planning for two dimensional robot motion," in *Proceedings IEEE/RSJ International Workshop on Intelligent Robots and Systems*, 1991, 818–823 vol.2.

[91] M. Bency, A. Qureshi, and M. Yip, "Neural path planning: Fixed time, near-optimal path generation via oracle imitation," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3965–3972, 2019.

[92] P. Adamu, J. Jegede, H. Okagbue, and P. Oguntunde, "Shortest path planning algorithm - a particle swarm optimization (PSO) approach," in *Proceedings of the World Congress on Engineering Vol I*, 2018, pp. 19–24.

[93] D. Agarwal and P. S. Bharti, "Matlab simulation of path planning and obstacle avoidance problem in mobile robot using SA, PSO and FA," *IEEE International Conference for Innovation in Technology (INOCON)*, pp. 1–6, 2020.

[94] M. Dorigo and M. Birattari, "Ant colony optimization," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 36–39.

[95] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Physica A: Statistical Mechanics and its Applications*, 2011.

[96] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy art: Fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Networks*, vol. 4, pp. 759–771, 1991.

[97] O. Semenova, A. Semenov, K. Koval, and A. Galka, "The ternary-encoded fuzzy-neural networks," in *Proceedings of International Conference on Modern Problem of Radio Engineering, Telecommunications and Computer Science*, 2012, pp. 305–305.

[98] Xiaochuan Wang and S. X. Yang, "A neuro-fuzzy approach to obstacle avoidance of a nonholonomic mobile robot," in *Proceedings IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, vol. 1, 2003, 29–34 vol.1.

[99] P. Malvido Fresnillo, S. Vasudevan, W. M. Mohammed, J. L. Martinez Lastra, and J. A. Perez Garcia, "Extending the motion planning framework - MoveIt with advanced manipulation functions for industrial applications," *Robotics and Computer-Integrated Manufacturing (RCIM)*, vol. 83, p. 102 559, 2023, ISSN: 0736-5845.

[100] D. Coleman, I. Sucan, S. Chitta, and N. Correll, *Reducing the barrier to entry of complex robotic software: A MoveIt! case study*, 2014. arXiv: `1404.3785 [cs.RO]`.

[101] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 4569–4574.

[102] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009, pp. 489–494.

[103] M. Likhachev, *Search-based planning library (SBPL)*, 2016.

[104] Stanford Artificial Intelligence Laboratory et al., *Robotic operating system*, version ROS Melodic Morenia, May 23, 2018. [Online]. Available: `https://www.ros.org`.

[105] Z. Kingston, M. Moll, and L. E. Kavraki, "Exploring implicit spaces for constrained sampling-based planning," *International Journal of Robotics Research*, vol. 38, no. 10–11, pp. 1151–1178, Sep. 2019.

[106] B. Cohen, I. A. Şucan, and S. Chitta, "A generic infrastructure for benchmarking motion planners," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 589–595.

[107] R. D. Párizs, D. Török, T. Ageyeva, and J. G. Kovács, "Machine learning in injection molding: An Industry 4.0 method of quality prediction," *Sensors*, vol. 22, no. 7, 2022, ISSN: 1424-8220.

[108] H. Jung, J. Jeon, D. Choi, and J.-Y. Park, "Application of machine learning techniques in injection molding quality prediction: Implications on sustainable manufacturing industry," *Sustainability*, vol. 13, no. 8, 2021, ISSN: 2071-1050.

[109] O. Ogorodnyk, O. V. Lyngstad, M. Larsen, K. Wang, and K. Martinsen, "Application of machine learning methods for prediction of parts quality in thermoplastics injection molding," in *Advanced Manufacturing and Automation VIII 8*, Springer, 2019, pp. 237–244.

[110] B. Silva, J. Sousa, and G. Alenya, "Machine learning methods for quality prediction in thermoplastics injection molding," in *International Conference on Electrical, Computer and Energy Technologies*, 2021.

[111] B. Silva, R. Marques, T. Santos, J. Sousa, and G. Alenya, "Relevant parameters identification in traditional & stretch and blow thermoplastics injection molding," in *9th International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications*, 2022.

[112] B. Silva, J. Sousa, and G. Alenya, "Data acquisition and monitoring system for legacy injection machines," in *IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications*, 2021.

[113] Z. Wang, C. Da Cunha, M. Ritou, and B. Furet, "Comparison of K-means and GMM methods for contextual clustering in HSM," *Procedia Manufacturing*, vol. 28, pp. 154–159, 2019, 7th International conference on Changeable, Agile, Reconfigurable and Virtual Production, ISSN: 2351-9789.

[114] D. Devaurs, T. Siméon, and J. Cortés, "Enhancing the transition-based RRT to deal with complex cost spaces," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 4120–4125.

[115] Í. Elguea-Aguinaco, A. Serrano-Muñoz, D. Chrysostomou, I. Inziarte-Hidalgo, S. Bøgh, and N. Arana-Arexolaleiba, "A review on reinforcement learning for contact-rich robotic manipulation tasks," *Robotics and Computer-Integrated Manufacturing (RCIM)*, vol. 81, p. 102 517, 2023.

[116] M. Suomalainen, Y. Karayiannidis, and V. Kyrki, "A survey of robot manipulation in contact," *Robotics and Autonomous Systems (RAS)*, vol. 156, p. 104 224, 2022.

[117] B. Nemec, R. Vuga, and A. Ude, "Exploiting previous experience to constrain robot sensorimotor learning," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2011, pp. 727–732.

[118] T. Yoshikawa, "Manipulability of robotic mechanisms," *The International Journal of Robotics Research*, vol. 4, pp. 3–9, 1985.

[119] N. Hansen, *The CMA evolution strategy: A tutorial*, 2016. arXiv: 1604.00772 [cs.LG].

[120] Z. Lončarević and A. Gams, "Accelerating robot reinforcement learning with accumulation of knowledge," in *International Conference on Robotics in Alpe-Adria Danube Region*, Springer, 2021, pp. 119–126.

[121] Z. Lončarević, A. Gams, and M. Simonič, "Source domain knowledge acquisition using simulated environment with minimized computing requirements," in *31st International Electrotechnical and Computer Science Conference*, 2022, pp. 228–231.

[122] K. Zaamout and J. Z. Zhang, "Improving neural networks classification through chaining," in *Artificial Neural Networks and Machine Learning*, A. E. P. Villa, W. Duch, P. Érdi, F. Masulli, and G. Palm, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 288–295.

[123] M. Volpp, F. Flürenbrock, L. Grossberger, C. Daniel, and G. Neumann, "Bayesian context aggregation for neural processes," in *International Conference on Learning Representations*, 2021.

Bibliography

Publications Related to the Thesis

Journal Articles

Zvezdan Lončarević, Andrej Gams, Simon Reberšek, Bojan Nemec, Jure Škrabar, Jure Skvarč, and Aleš Ude. Specifying and optimizing robotic motion for visual quality inspection. *Robotics and Computer-Integrated Manufacturing*, 72:102200, 2021

Rok Pahič, Zvezdan Lončarević, Andrej Gams, and Aleš Ude. Robot skill learning in latent space of a deep autoencoder neural network. *Robotics and Autonomous Systems*, 135:103690, 2021

Conference Paper

Zvezdan Lončarević, Ge Li, Gerhard Neumann, and Andrej Gams. An encoder-decoder architecture for smooth motion generation. In Tadej Petrič, Aleš Ude, and Leon Žlajpah, editors, *Advances in Service and Industrial Robotics*, pages 358–366, Cham, 2023. Springer Nature Switzerland

Zvezdan Lončarević, Simon Reberšek, Aleš Ude, and Andrej Gams. Randomized robotic visual quality inspection with in-hand camera. In Ivan Petrovic, Emanuele Menegatti, and Ivan Marković, editors, *Intelligent Autonomous Systems 17*, pages 483–494, Cham, 2023. Springer Nature Switzerland

Zvezdan Lončarević, Tadej Petrič, and Andrej Gams. Fitting constrained trajectory with high variability into redundant robot workspace. In Andreas Müller and Mathias Brandstötter, editors, *Advances in Service and Industrial Robotics*, pages 167–175, Cham, 2022. Springer International Publishing

Zvezdan Lončarević, Mihael Simonič, Aleš Ude, and Andrej Gams. Combining reinforcement learning and lazy learning for faster few-shot transfer learning. In *21st International Conference on Humanoid Robots (Humanoids)*, pages 285–290, 2022

Zvezdan Lončarević, Andrej Gams, and Mihael Simonič. Source domain knowledge acquisition using simulated environment with minimized computing requirements. In *31st International Electrotechnical and Computer Science Conference*, pages 228–231, 2022

Zvezdan Lončarević, Aleš Ude, and Andrej Gams. Accelerated robot skill acquisition by reinforcement learning-aided sim-to-real domain adaptation. In *20th International Conference on Advanced Robotics (ICAR)*, pages 269–274, 2021

Zvezdan Lončarević and Andrej Gams. Accelerating robot reinforcement learning with accumulation of knowledge. In *International Conference on Robotics in Alpe-Adria Danube Region*, pages 119–126. Springer, 2021

Zvezdan Lončarević, Rok Pahič, and Andrej Gams. Learning of the velocity profile for quality inspection motion using PoWER. In *29th International Electrotechnical and Computer Science Conference*, pages 156–159, 2020

Zvezdan Lončarević, Rok Pahič, Mihael Simonič, Aleš Ude, and Andrej Gams. Reduction of trajectory encoding data using a deep autoencoder network: robotic throwing. In *International Conference on Robotics in Alpe-Adria Danube Region*, pages 86–94. Springer, 2019

Rok Pahič, Zvezdan Lončarević, Aleš Ude, Bojan Nemec, Andrej Gams, et al. User feedback in latent space robotic skill learning. *In 18th International Conference on Humanoid Robots (Humanoids)*, pages 270–276. IEEE, 2018

Other Publications

Journal Articles

Zvezdan Lončarević, Rok Pahič, Aleš Ude, and Andrej Gams. Generalization-based acquisition of training data for motor primitive learning by neural networks. *Applied Sciences*, 11(3):1013, 2021

Conference Paper

Zvezdan Lončarević and Andrej Gams. Using neural networks for synthesizing importance sampler database in reinforcement learning. In *30th International Electrotechnical and Computer Science Conference*, pages 179–182, 2021

Zvezdan Lončarević, Rok Pahič, Mihael Simonič, Aleš Ude, and Andrej Gams. Generalization based database acquisition for robot learning in reduced space. *In International Conference on Robotics in Alpe-Adria Danube Region*, pages 496–504. Springer, 2020

Zvezdan Lončarević, Rok Pahič, Gregor Papa, and Andrej Gams. Experimental evaluation of deep-learning applied on pendulum balancing. In *28th International Electrotechnical and Computer Science Conference*, pages 219–222, 2019

Zvezdan Lončarević, Mihael Simonič, Aleš Ude, Andrej Gams, et al. Learning of robotic throwing at a target using a qualitative learning reward. *In EUROCON 2019-18th International Conference on Smart Technologies*, pages 1–6., 2019

Zvezdan Lončarević, Rok Pahič, Aleš Ude, Bojan Nemec, and Andrej Gams. Replacing reward function with user feedback. In *27th International Electrotechnical and Computer Science Conference*, pages 139–142, 2018

Zvezdan Lončarević and Andrej Gams. Skill adaptation using transfer learning. In *14th Jožef Stefan International Postgraduate School Students' Conference*, page 62, 2022

Zvezdan Lončarević and Andrej Gams. Generation of a priori knowledge for reinforcement learning using neural networks. In *13th Jožef Stefan International Postgraduate School Students' Conference and 15th Young Researchers' Day of Chemistry, material science, biochemistry and environment*, page 46, 2021

Zvezdan Lončarević and Andrej Gams. Robotic camera motion optimization for minimal image quality degradation. In *12th Jožef Stefan International Postgraduate School Students' Conference and 14th Young Researchers' Day*, page 35, 2020

Zvezdan Lončarević, Rok Pahič, Mihael Simonič, and Andrej Gams. Human intuitive reward systems for reinforcement learning of robotic actions in latent space of deep auto encoder network. *In 11th Jožef Stefan International Postgraduate School Students' Conference and 13th Young Researchers' Day*, page 45, 2019

Biography

Education

1.10.2017 - 7.10.2019: **M.Sc. in Information and Communication Technologies**
Jožef Stefan International Postgraduate School
Final Average grade 9.75
Thesis Title: "Robotic Learning in Feature Space of Deep Autoencoders"
1.10.2013 - 13.7.2017: **B.Sc. in Mechatronics**
Faculty of Technical Science, University of Novi Sad
Final Average grade 9.28
Thesis Title: "Analysis of Lateral Dynamics of Vehicle"

Conferences

June 2023: Attended with a contribution (article and oral presentation) at the *"32nd International Conference on Robotics in Alpe-Adria-Danube Region"*, Bled, Slovenia – Title of the presentation: "An Encoder-Decoder Architecture for Smooth Motion Generation"
June 2022: Attended with a contribution (article and oral presentation) at the *"31st International Conference on Robotics in Alpe-Adria-Danube Region"*, Klagenfurt, Austria – Title of the presentation: "Fitting constrained trajectory with high variability into redundant robot workspace"
June 2022: Attended with a contribution (abstract, poster, elevator pitch) at the *"14th Jožef Stefan International Postgraduate School Students' Conference"*, Kamnik, Slovenia – Title of the presentation: "Skill adaptation using transfer learning"
December 2021: Attended with a contribution (article and oral presentation) at the *"20th International Conference on Advanced Robotics"*, Ljubljana, Slovenia – Title of the presentation: "Accelerated robot skill acquisition by reinforcement learning-aided sim-to-real domain adaptation"
September 2021: Attended with a contribution (article and oral presentation) at the *"30th International Electrotechnical and Computer Science Conference"*, Portorož, Slovenia – Title of the presentation: "Using neural networks for synthesizing importance sampler database in reinforcement learning"
June 2021: Attended with a contribution (article and oral presentation) at the *"30th International Conference on Robotics in Alpe-Adria-Danube Region"*, Poitiers, France – Title of the presentation: "Accelerating robot reinforcement learning with accumulation of knowledge"
May 2021: Attended with a contribution (abstract, poster, elevator pitch) at the *"13th Jožef Stefan International Postgraduate School Students' Conference"* and *"15th Young Researchers' Day of Chemistry, material science, biochemistry and environment"*, Ljubljana, Slovenia – Title of the presentation: "Generation of a priori knowledge for reinforcement learning using neural networks"

September 2020: Attended with a contribution (article and oral presentation) at the *"29th International Electrotechnical and Computer Science Conference"*, Portorož, Slovenia – Title of the presentation: "Learning of the velocity profile for quality inspection motion using PoWER"
September 2019: Attended with a contribution (article and oral presentation) at the *"28th International Electrotechnical and Computer Science Conference"*, Portorož, Slovenia – Title of the presentation: "Experimental evaluation of deep-learning applied on pendulum balancing"
June 2019: Attended with a contribution (article and oral presentation) at the *"28th International Conference on Robotics in Alpe-Adria-Danube Region"*, Kaiserslautern, Germany – Title of the presentation: Reduction of trajectory encoding data using a deep autoencoder network: robotic throwing
April 2019: Attended with a contribution (abstract, poster, elevator pitch) at the *"11th Jožef Stefan International Postgraduate School Students' Conference"* and *"13th CMBE day"*, Planica, Slovenia – Title of the presentation: "Human intuitive reward systems for reinforcement learning of robotic actions in latent space of deep autoencoder network"
September 2018: Attended with a contribution (article and oral presentation) at the *"27th International Electrotechnical and Computer Science Conference"*, Portorož, Slovenia – Title of the presentation: "Replacing reward function with user feedback"

Research Visits

August 2022 – January 2023 (6 months): Exchange to German Karlsruhe Institute of Technology (Autonomous learning robots laboratory) funded by German Academic Exchange Programme (DAAD)

Memberships

October 2021 - September 2022: *Jožef Stefan International Postgraduate School* student council member

Permissions

This thesis contains text excerpts and figures from conference papers and journal articles published during the doctoral studies.