TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

ACIN
AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Improving Industrial Robot Accuracy utilizing Iterative Learning Control with Visual Environment Feedback

DIPLOMARBEIT

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Diplom-Ingenieurs (Dipl.-Ing.)

unter der Leitung von

Univ.-Prof. Dr.sc.techn. Georg Schitter
Univ.-Ass. Dipl.-Ing. Peter Gsellmann
Univ.-Ass. Dipl.-Ing. Christoph Buchner

eingereicht an der
Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik
Institut für Automatisierungs- und Regelungstechnik

von

Niklas Paternoster
Matrikelnummer: 12110175

Wien, im März 2025

Technische Universität Wien
Karlsplatz 13, 1040 Wien, Österreich

Abstract

Collaborative robot enable direct human-robot interaction in shared workspaces, offering space efficiency and enhanced safety through lightweight components. However, these lighter and therefor less rigid components introduce significant trajectory deviations and accuracy limitations at the Tool Center Point (TCP). This thesis proposes a vision-based Iterative Learning Control (ILC) approach to compensate for both systematic kinematic and non-kinematic errors in repetitive tasks.
The proposed approach follows a two-phase workflow: establishing ground truth and updating the trajectory. During the ground truth phase, images captured by an eye-in-hand camera are stored and used to construct a 3D map through triangulation. In the update phase, newly captured images are processed using an optimization-based monocular Visual Odometry (VO) algorithm to estimate deviations in the camera's movement. Based on these deviations, a PD-type ILC iteratively refines the trajectory, improving accuracy over successive executions. To achieve this, the computed TCP deviations are converted into the required joint adjustments using inverse kinematics and the robot's kinematic model.

A simulation evaluation was conducted to assess the VO algorithm's performance. The study examines how environmental conditions and camera selection affect the recognition of camera movement. The results show that noise, 3D depth data, and feature count significantly influence the detectable Signal-to-Noise Ratio (SNR) when used as an ILC input. Even with a noise standard deviation of 1 pixel and only 20% depth data availability, the SNR remains above 1, enabling the detection of deviations of 0.5 mm and 0.5 mrad.

Further real-world experiments confirm these findings. For a given trajectory, with dynamic turning points and an added flexible end-effector setup the proposed approach effectively identifies and corrects both kinematic and non-kinematic errors in the end-effector's positioning. It successfully reduces the Absolute Trajectory Error (ATE) of 0.25 m and 0.3 rad to below 3 mm and 5 mrad over the course of 100 iterations. The proposed framework thus enables high-precision motion for predefined tasks, such as repetitive pick-and-place applications, even with flexible error-prone robot components.

Kurzfassung

Collaborative robots (cobots) ermöglichen eine enge Zusammenarbeit zwischen Mensch und Maschine in gemeinsamen Arbeitsbereichen, wodurch sie Platz sparen und dank ihrer leichten Bauweise ein höheres Maß an Sicherheit bieten. Allerdings führen diese leichteren, weniger starren Komponenten zu erheblichen Abweichungen der Trajektorie am TCP des Roboters.
Diese Arbeit präsentiert ein kamerabasiertes ILC-System, das systematische kinematische und nicht-kinematische Fehler des Roboterendeffektors bei sich wiederholenden Aufgaben kompensiert. Das System verwendet eine am Endeffektor montierte Ka- mera und arbeitet in zwei Phasen: In der Ground-Truth-Phase werden Referenzbilder aufgenommen und mithilfe von Triangulation eine 3D-Karte erstellt. Anschließend werden die Bilder mit der fehlerhaften Trajektorie aufgenommen und mittels einer optimierungbasierenden visuellen Odometrie (VO) analysiert. Dabei werden Bewegungsab- weichungen der Kamera und somit des Endeffektors im Vergleich zu den Referenzbildern ermittelt. Auf Basis dieser erfassten Abweichungen optimiert ein PD-basierter ILC-Prozess die Robotertrajektorie, indem er gezielte Anpassungen an den Gelenkwinkeln vornimmt. Dabei werden die berechneten TCP-Abweichungen mithilfe inverser Kinematik in die erforderlichen Gelenkwinkel umgewandelt.

Simulationen bestätigen die Robustheit des VO-Algorithmus unter verschiedenen Umgebungsbedingungen und Kameraeinstellungen. Die Ergebnisse zeigen, dass Faktoren wie Bildrauschen, 3D-Tiefendaten und die Anzahl der erkannten Merkmale das Signal-Rausch-Verhältnis (SNR) beeinflussen. Selbst bei einem Bildrauschen mit einer Standardabweichung von 1 Pixel und einer Tiefendatenverfügbarkeit von nur 20% können Abweichungen von 0.5 mm und 5 mrad zuverlässig erfasst werden.
Experimente am Roboter unter Verwendung einer Trajektorie mit dynamischen Wendepunkten und einem flexiblen Endeffektor-Setup belegen die Effektivität des Systems. Die absolute Trajektorienabweichung (ATE) wird so innerhalb von 100 Iterationen von 0.25 m und 0.3 rad auf unter 3 mm und 5 mrad reduziert.
Das vorgeschlagene System ermöglicht so präzise Pick-and-Place-Aufgaben selbst mit flexiblen Roboterkomponenten.

Contents

Nomenclature

Δx	Difference of change in x
$x_{\max}$	Maximum value of x
$x_{\min}$	Minimum value of x
$\hat{x}$	Estimated or predicted value of x
$\mathbf{M}$	Matrix
$\mathbf{P}_i^j$	Pose at waypoint i for trajectory j
$\bar{\mathbf{v}}$	Normalized vector
$\mathbf{v}$	Vector

Acronyms

FLANN Fast Library for Approximate Nearest Neighbors.

ROS Robot Operating System.

2D Two-dimensional.

3D Three-dimensional.

ATE Absolute Trajectory Error.

cobots collaborative robots.

DH Denavit-Hartenberg.

FoV Field of View.

ILC Iterative Learning Control.

kNN k-Nearest Neighbours.

LM Levenberg–Marquardt algorithm.

PD-Type Proportional-Derivative Type.

PoE Product of Exponentials.

RANSAC Random Sample Consensus.

RE Relative Error.

SfM Structure from Motion.

SIFT Scale-Invariant Feature Transform.

SNR Signal-to-Noise Ratio.

SVD Singular Value Decomposition.

TCP Tool Center Point.

VO Visual Odometry.

List of Tables

List of Figures

CHAPTER 1

Introduction

The emergence of Industry 4.0 has accelerated the market's need for automation and interconnected industrial processes. This progression towards more intelligent manufacturing systems has exposed substantial limitations in conventional automation concepts, particularly with regard to the interaction between humans and robots, and integration into existing systems. [1]
Although industrial robots have proven effective for automating production lines, particularly in repetitive tasks, their operation is limited to areas isolated from human workers due to safety concerns. The forces required to manipulate heavy objects pose lethal risks to humans, necessitating strict separation between robots and workers. Implementing these systems into existing production processes and facilities presents significant challenges, and often requires extensive changes to existing workflows and infrastructure.
These systems require extensive safeguarded workspace and, while programmable for various tasks, lack the real-time adaptability required in modern manufacturing environments [2]. In order to integrate robots into an existing production line, process steps often need to be modified and adapted to the robot application. These major changes to the process line involve substantial investments, which can be difficult to justify financially, especially for medium-sized companies.

Collaborative robots (cobots) address this integration limitation by enabling direct human-robot interaction within a shared workspace. This shared workspace not only saves valuable floor space but enables effective collaboration, where robots handle physically demanding operations while humans focus on complex, variable tasks, as it can be seen in Figure 1.1. In order for cobots to be able to work hand in hand with human workers, and simultaneously ensure human safety in the event of an unavoidable collision, they are often built with lighter and more flexible components than traditional industrial robots, reducing both manufacturing costs and the risk of injury from accidental human contact.

However, the utilization of these more flexible components results in deviations between the robot's intended task and its actual performance, as well as its designated trajectory. These deviations impact the Tool Center Point (TCP), which denotes the operational point of the robot's end effector where the designated task is executed.
These inaccuracies originate from errors, introduced by additional joint elasticities,

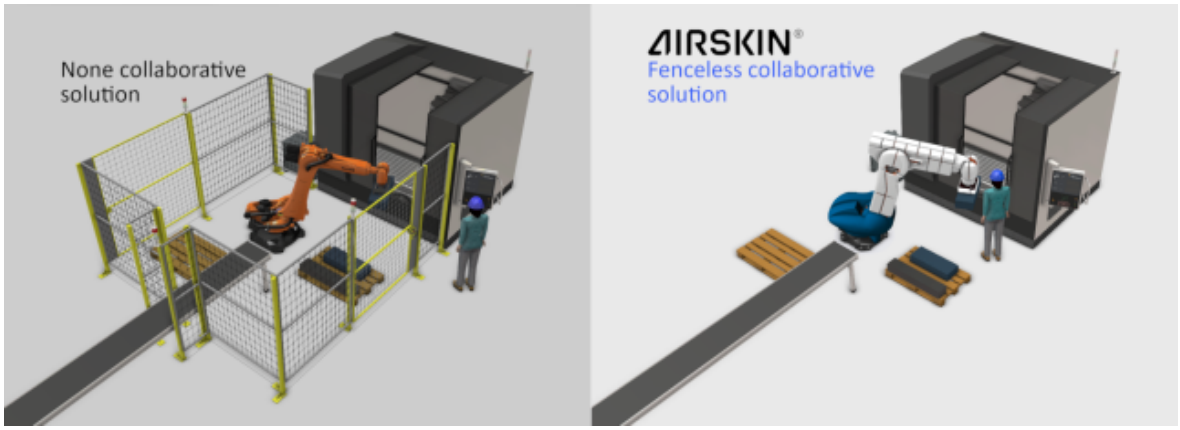


Figure 1.1: Comparison of an industrial robot inside a predefined workspace guarded by a fence on the left side and a collaborative robot solution without any area restrictions for the human worker on the right side. [3]

friction, transmission errors, and other kinematic and non-kinematic errors.
Additional sensor systems are used to mitigate these newly introduced inaccuracies by using multiple different approaches such as force or torque sensors at the joints, vision-based tracking systems for position correction, high-precision encoders for joint angle measurements, and real-time feedback control algorithms that can compensate for elastic deformation and transmission errors [4–6].
A relatively inexpensive and frequently used sensor in a robotic system is a camera system. The camera is installed either as an external observer of the structure or directly on the robot's TCP. The mounting on the TCP is referred to as an eye-in-hand configuration and, due to its position, directly detects all intentional and unintentional movements of the robot (cf. Figure 1.2).

These adapted robotic systems are typically controlled by model-based closed-loop control strategies that use feedback from the various sensors to continuously correct deviations from the desired trajectory. Although these conventional control approaches can compensate for some of the above inaccuracies, they often struggle to completely eliminate systematic errors that occur during task execution, especially at the beginning of a trajectory. However, many industrial applications, such as assembly, pick-and-place, or material handling, are characterized by their repetitive nature. This repetitive nature of robotic tasks provides an opportunity for more advanced control strategies.
The Iterative Learning Control (ILC) utilize repetitive tasks by acquiring knowledge from prior executions to enhance subsequent task performance. This process enables the control system to directly compensate for systematic errors, such as those from kinematic and dynamic flexing in the links and joints, through the iterative repetition of the task trajectory and the subsequent correction of these errors.

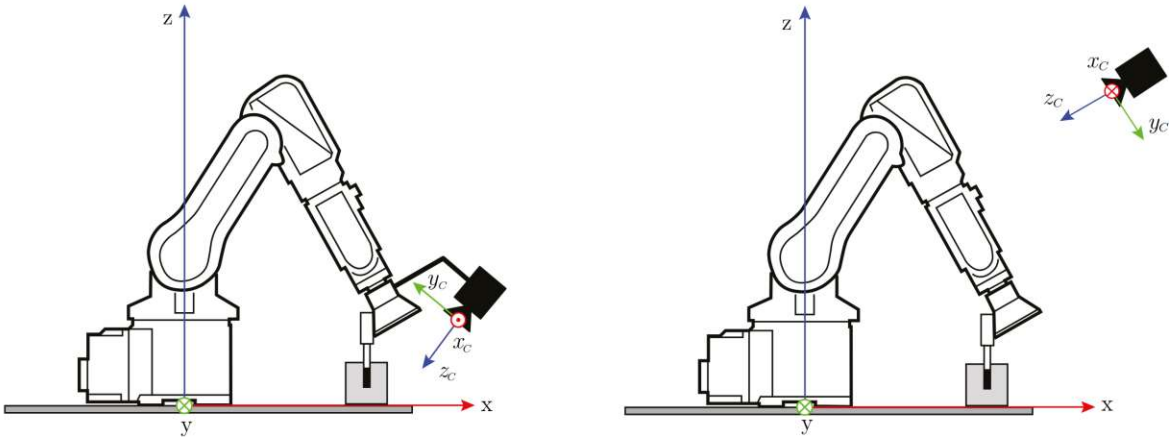


Figure 1.2: Representation of an eye-in-hand camera configuration on the left side, where the camera is mounted onto the robot end effector. On the right side, the camera is externally mounted with a fixed relationship to the world reference frame. [7]

Combining the additional sensors with the ILC's control concept and using the environment of the robot provides a novel approach to compensating for the errors introduced by the more flexible cobots.

1.1 Scope of Thesis

This work integrates ILC with an eye-in-hand camera system for iterative error compensation for repeating tasks. The camera acts as the sole sensor for error detection, enabling the system to identify and iteratively correct both kinematic and non-kinematic deviations. This motion error detection is used to update the trajectory until the desired accuracy is met.

1.2 Outline of Thesis

This thesis is organized as follows: Chapter 2 reviews state-of-the-art methods in industrial robotics, vision-based systems, and ILC approaches. Chapter 3 details the implementation of the motion error estimation and ILC architecture, including sensor integration and control algorithms. Chapter 4 validates the proposed approach through both simulation studies and real-world experiments on an adapted industrial robot. Chapter 5 finishes with a comprehensive analysis of the results, system limitations, and potential improvements for vision-based ILC in collaborative robotics applications.

CHAPTER 2

State of the Art

In this chapter, the fundamentals for the proposed vision-based Iterative Learning Control (ILC) concept are described. The chapter delves into the essential aspects of robotics and machine vision, beginning with an overview of robots and their classification. Subsequently, the forward and inverse kinematics problem of a robot are presented, followed by methods for planning its trajectory. Following this, the ILC concept is explained in greater detail. Thereafter, some key principles of machine vision, with a focus on visual odometry, are discussed more comprehensively. Finally, the Robot Operating System (ROS) middleware and its communication patterns are presented.

2.1 Robots

A robot is a programmable machine built with the aim to carry out a series of actions autonomously or semi-autonomously [8]. It may be classified by its capability to operate in different environments and its related mobility, as described in Figure 2.1 [9]. A robot can, therefore, be divided into two categories: the fixed robot and the mobile robot. Mobile robots can be further subdivided into the environments in which they operate. Furthermore, robots can be divided into industrial and service robots based on their

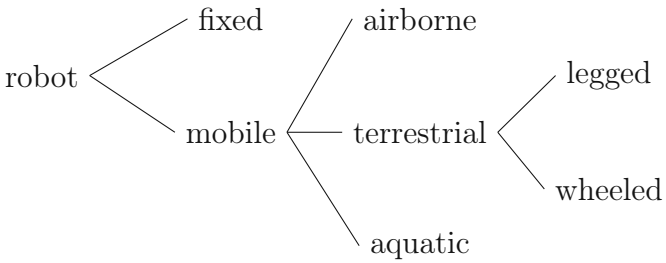


Figure 2.1: Classification of robots by environment interaction [9].

task field and application. Service robots operate in fields such as medical assistance

[10], domestic chores [11], and human transportation such as self-driving cars [12].
Industrial robots often operate in enclosed environments and are primarily used for tasks such as mechanical manufacturing operations, assembly, or logistics operations [13]. They are defined as an automatically controlled, reprogrammable multipurpose manipulator with three or more programmable axes, including a chain of rigid bodies and its actuators, and the robot controller [8].
Industrial robots are characterized by their mechanical configuration, which is essential for understanding their movement capabilities and effectively controlling their operations. This configuration typically involves multiple rigid segments called links which are connected by joints [8], allowing specific movements while limiting others. Various types of joints are used in industrial robots, including rotary joints, prismatic joints, and spherical joints, each providing different movement capabilities.
Certain segments of the robot are assigned more specialized functions, such as the base, which secures the robot to the ground or a platform, and the end-effector, which is fixed to the last link of the robot arm and serves as the device interacting with the environment [14]. A crucial aspect of the end-effector's functionality is the Tool Center Point (TCP), a precise reference point that helps determine the end-effector movements and ensures accuracy in tasks such as picking, placing, welding, or painting.
The entire structure of links and joints is called a kinematic chain and grants the robot a specific range of motion and degrees of freedom. Based on the combination of links and joints inside this chain, the robot is considered either a serial or parallel kinematic structure. The difference between these two kinematic configurations is that in a parallel kinematic chain, multiple links connect to a single joint, creating a closed-loop system, as illustrated in Figure 2.2. In a serial kinematic chain, on the other hand, the links are connected end-to-end, forming an open-loop structure where each joint connects two links with an end-effector at the end.

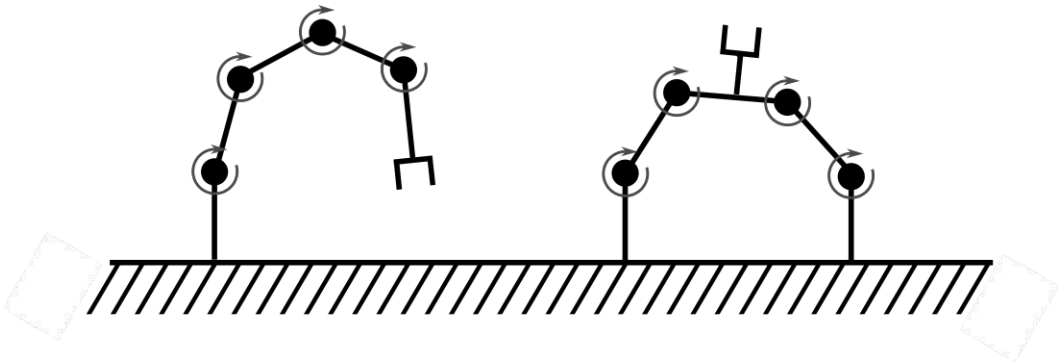


Figure 2.2: Illustration of two different robot configurations. On the left side, a serial kinematic chain is shown, in which the rotary joints are represented as small spheres connected by lines indicating the links. The configuration on the right shows a parallel kinematic chain. In both kinematic chains, the end-effector is illustrated as a clamp.

2.2 Kinematics

In robotics, kinematics involves the analytical study of the position and orientation changes, also called motion, of an end-effector without considering the forces and moments that drive these changes [15]. The consideration of these forces and moments falls under the study of robot dynamics, which are not further considered in this thesis.

2.2.1 Homogeneous Transformation Matrix

To describe these changes of position and orientation in a Three-dimensional (3D) space, homogeneous transformation matrices can be used. A homogeneous transformation matrix combines rotation and translation into a single matrix operation, which simplifies the coordinate transformation from one reference frame to another [16].

$$\mathbf{T}_{\mathcal{C}}^{\mathcal{W}} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{2.1}$$

The homogeneous transformation in Equation (2.1), describes the pose of the coordinate system $\mathcal{C}$ in relation to the inertial coordinate system $\mathcal{W}$, with $\mathbf{R} \in SO(3)$ being the rotation matrix and $\mathbf{t} \in \mathbb{R}^3$ representing the translation vector.
For the rotation matrix $\mathbf{R}$, a rotation can be parameterized in multiple ways, including Euler angles and quaternions [14]. Euler angles represent rotations about the axes of a coordinate system, typically defined with three angles $\phi = [\varphi\ \theta\ \psi]^T \in \mathbb{R}^3$. The orientation change is performed by subsequent rotation with these angles around the coordinate axis $i, j, k \in \{x, y, z\}$. An example of a rotation done with Euler angles using the axis order *z-x-z'* is visualized in Figure 2.3. However, Euler angles can suffer from gimbal lock, which describes a specific set of angles ϕ, where one rotational degree of freedom is lost. This leads to a reduced capability to describe the update in the orientation of these, making them less suitable for certain applications.
Unit quaternions, which are four-dimensional complex numbers, avoid gimbal lock and provide a reliable and continuous representation of rotation, making them highly suitable for precise orientation tasks. The use of unit quaternions in robotics allows for smooth interpolation between orientations, with the drawback of being less intuitive than the Euler angles. [14, 17]

Understanding the pose of the end-effector through these transformations is crucial for describing it in space. This leads to the concepts of forward and inverse kinematics, which are fundamental for determining the end-effector's pose based on joint parameters and vice versa.

2.2.2 Forward Kinematics

Forward kinematics involves calculating the position and orientation of the end-effector with given joint parameters in a kinematic chain relative to the robot's base frame. This is achieved by utilizing the known joint and link parameters, establishing a direct mapping from the robot's configuration space to its Cartesian space. If the absolute coordinates of the robot base are known, the position and orientation of the end-effector

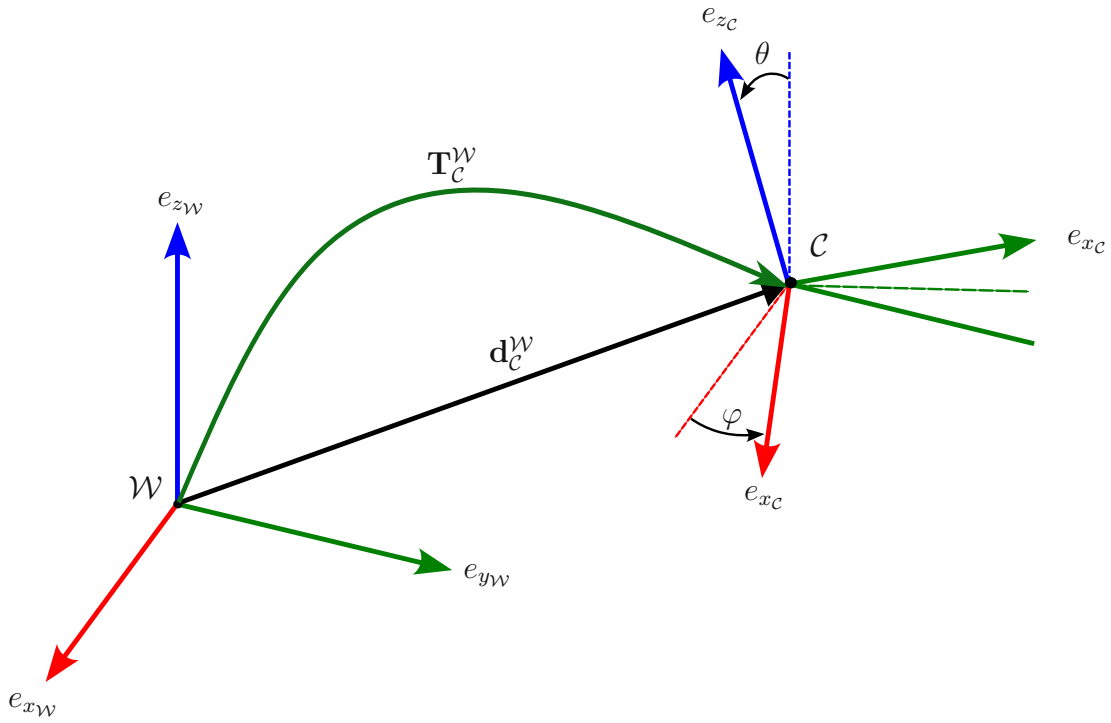


Figure 2.3: Representation of inertial coordinate system $\mathcal{W}$ and the coordinate system $\mathcal{C}$, where $\mathcal{C}$ is shifted by the translation vector $\mathbf{d}_\mathcal{C}^\mathcal{W}$ and rotated by the z-x-z" Euler angles φ and θ and a non zero-valued ψ.

can also be expressed in absolute world coordinates.

Focusing entirely on a serial kinematic chain, a widely used method for formulating the forward kinematics with only rotary and linear joints, is the Denavit-Hartenberg (DH) convention [18]. The DH convention simplifies the process of modeling robotic kinematics by standardizing the coordinate frames and transformations. Thus, to obtain such simplifications, it is necessary to associate each joint i of the robot with a coordinate frame, and express the relationship between successive frames using four parameters: link length a_i, link twist α_i, link offset d_i, and joint angle θ_i.
The transformation from one frame to the next frame with a given $q_i \in \{\theta_i, d_i\}$ is represented by the homogeneous transformation matrix

$$\mathbf{T}_i^{i-1}(q_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.2}$$

The input q_i describes either the rotation angle in case joint i is a rotary joint, or the translation value d_i in case joint i is a prismatic joint.
By sequentially multiplying these transformation matrices from Equation (2.2) together, the pose of the end-effector relative to the base frame can be determined. For a serial n-joint robot, the overall transformation matrix from the base frame to the end-effector

is

$$\mathbf{T}_n^0 = \mathbf{T}_1^0 \mathbf{T}_2^1 \mathbf{T}_3^2 \cdots \mathbf{T}_n^{n-1}. \tag{2.3}$$

Another more modern approach to forward kinematics is the Product of Exponentials (PoE) method, which takes advantage of the properties of Lie groups and Lie algebras [19]. In the PoE method, the forward kinematics of a robot arm is expressed as a product of exponential maps of twists, which describe the motion of each joint.
PoE, specifically the space form of the PoE formula for a robot with n-joints is defined as

$$\mathbf{T}(\theta) = e^{[\mathbf{S}_1]\theta_1} e^{[\mathbf{S}_2]\theta_2} \cdots e^{[\mathbf{S}_n]\theta_n} \mathbf{M}, \tag{2.4}$$

where $\mathbf{S}_k$ is the screw axis vector for joint k, consisting of an angular velocity vector $\boldsymbol{\omega} \in \mathbb{R}^3$ and a linear velocity vector $\mathbf{v} \in \mathbb{R}^3$. The bracket notation $[\mathbf{S}_k]$ denotes the matrix representation in SE(3) as $[\mathbf{S}_k] = \begin{bmatrix} [\boldsymbol{\omega}] & \mathbf{v} \\ 0 & 0 \end{bmatrix}$, where $[\boldsymbol{\omega}]$ is the skew-symmetric matrix form of the angular velocity vector. The joint displacement value is represented by θ_k and $\mathbf{M}$ is the configuration of the end-effector in the zero position. [20]

2.2.3 Inverse Kinematics

The inverse kinematics involves determining a joint configuration that achieves a given end-effector pose in Cartesian space. Unlike forward kinematics, which has a straightforward and general solution, inverse kinematics does not offer a universally applicable solution for any serial kinematic chain [21].
This challenge is particularly pronounced in inverse kinematics due to several factors. The equations involved are usually non-linear, making it difficult to find closed-form solutions. In addition, there may be multiple or even infinite solutions, especially for redundant robots that have more degrees of freedom than required for a given task [14, 22]. To address these complexities, various methods have been developed, each suitable for different applications.
For simple robots with few degrees of freedom or manipulators with specially designed kinematics, a closed-form analytical solution can be used, providing explicit equations for the joint parameters [14, 23, 24].

Numerical methods encompass iterative approaches that aim to find a satisfactory solution by minimizing the error of the underlying non-linear system. These methods can be broadly categorized into three types: *Jacobian*, *Newton*, and *Heuristic* methods [21].

Jacobian Methods

Jacobian methods use the Jacobian matrix $\mathbf{J}$, which is a matrix of partial derivatives of the system with respect to the joint parameters $\boldsymbol{\theta}$. The Jacobian matrix relates the joint velocities to the end-effector velocities (cf. Equation (2.6))[25]. The basic approach involves iteratively updating the joint parameters to minimize the error between the current and desired end-effector positions. The entries of the Jacobian matrix are

defined as

$$\mathbf{J}(\boldsymbol{\theta})_{ij} = \left(\frac{\partial x_i}{\partial \theta_j} \right), \tag{2.5}$$

where $\boldsymbol{\theta} = (\theta_1, ..., \theta_n)^T$ with $i \in [1, k]$ and $j \in [1, n]$. Here, k represents the number of end-effectors, and n represents the number of joints.
The basic equation for forward dynamics that describes the velocities of the end-effectors can be written as follows, using the dot notation for time derivatives:

$$\dot{\boldsymbol{x}} = \mathbf{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}. \tag{2.6}$$

The goal is to minimize the error $\mathbf{e}$ for a value $\boldsymbol{\theta}$, between the actual end-effector position $\boldsymbol{x}(\boldsymbol{\theta})$ and the desired target end-effector position $\mathbf{t}$:

$$\mathbf{e} = \mathbf{t} - \boldsymbol{x}(\boldsymbol{\theta}). \tag{2.7}$$

To solve this, small joint angle updates $\Delta\boldsymbol{\theta}$ are iteratively applied to an initial value. The relationship between changes in joint angles and the resulting change in end-effector position $\Delta\boldsymbol{x}$ is approximated using the Jacobian:

$$\Delta\boldsymbol{x} \approx \mathbf{J}\Delta\boldsymbol{\theta}. \tag{2.8}$$

The required joint angle update $\Delta\boldsymbol{\theta}$ can then be estimated using the inverse of the Jacobian:

$$\Delta\boldsymbol{\theta} \approx \mathbf{J}^{-1}\mathbf{e}. \tag{2.9}$$

The final joint angles for a desired end-effector position are obtained by iteratively updating $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}$ until the error $\mathbf{e}$ falls below a desired threshold or a maximum number of iterations is reached.
Due to potential non-invertibility, non-square nature, and the presence of singularities in the Jacobian matrix, different methods like the Jacobian Transpose, Jacobian Pseudo-Inverse, damped least-squares [26] or Singular Value Decomposition (SVD) [27] are applied.

Newton Methods

Newton methods are based on a second-order Taylor expansion of the objective function. These methods utilize the Hessian matrix to find a second-order approximation of the inverse kinematics problem. However, calculating the Hessian is computationally expensive. The quasi-Newton methods, like BFGS, approximate the Hessian to reduce computational costs while avoiding singularity issues [28].

Heuristic Methods

Heuristic methods employ simpler, iterative approaches that do not rely on complex equations [21]. These methods are generally faster and computationally cheaper, but may not produce biomechanically feasible motions. A common method is Cyclic Coordinate Descent (CCD), which iteratively adjusts one joint at a time to reduce the end-effector error, starting inward from the end-effector to the base [29, 30]. Another

method is the Forward and Backward Reaching Inverse Kinematics (FABRIK), which updates joint positions iteratively by moving forward and backward along the chain to minimize the distance to the target [31].

2.3 Robot Motion Planning

In robotics, planning a robot's motion is a fundamental task that involves determining a feasible path for the robot to move from one configuration to another, avoiding obstacles and complying with its physical constraints [14]. This path is subsequently extended with a time parameterization, resulting in a trajectory. This section will cover the basics of path planning and then expand into the area of trajectory planning.

2.3.1 Path Planning

Path planning addresses the problem of determining a feasible path for the robot to move through its environment with potential obstacles. This involves not only finding a valid path, satisfying the start and end configuration, but also ensuring that the path is collision-free, respects the robot's kinematic and dynamic constraints, and is computationally efficient [20].
The complexity of the path can vary, from a simple movement between two points of the end-effector in space to a specific constructed path with multiple obstacles and constraints.
The path planning is carried out in the configuration space $\boldsymbol{\theta}(s)$, whereas end-effector movements are planned in Cartesian space $\boldsymbol{x}_e(s)$. The variable $s \in [0, 1]$ describes here the progress of the path, where $s = 0$ is the starting position and $s = 1$ the end position of the robot.
To effectively overcome these challenges, various approaches have been developed, among which sampling-based methods have proven to be particularly successful. Two prominent examples of these methods are Rapidly Exploring Random Trees (RRT) and Probabilistic Roadmap (PRM). RRT is an algorithm that incrementally builds a search tree by randomly sampling the configuration space and extending the tree towards the sampled points [32]. The PRM algorithm generates a roadmap R by randomly selecting nodes from the configuration or state space and adding them to the roadmap based on the shortest distance [33].

2.3.2 Trajectory Planning

Building upon the paths generated in the previous section, represented as $\boldsymbol{\theta}(s)$ or $\boldsymbol{x}_e(s)$, the next step involves planning the trajectory, where the focus shifts to the robot's movement over time. To achieve this, the time parameterization $s(t)$ is added, assigning a value s for each time $t \in [0, T]$, $s : [0, T] \to [0, 1]$.
The resulting trajectory $\boldsymbol{\theta}(s(t))$ or $\boldsymbol{x}_e(s(t))$ should then be sufficiently smooth for usage with the robot's analog drives and controller, but also respect the permitted position, speed, and acceleration limits of the actuator.
Time parameterization of a simple point-to-point path can be accomplished by using

methods like polynomial-time scaling or trapezoidal motion profiles. Polynomial functions offer smooth transitions but may require complex calculations to solve for the constraints, while trapezoidal motion profiles provide a straightforward approach with easily controlled acceleration and deceleration phases, though they may produce less smooth motion at transitions. A basic representation of the motion profile of both the trapezoid and the polynomial-time scaling can be seen in Figure 2.4 and Figure 2.5.

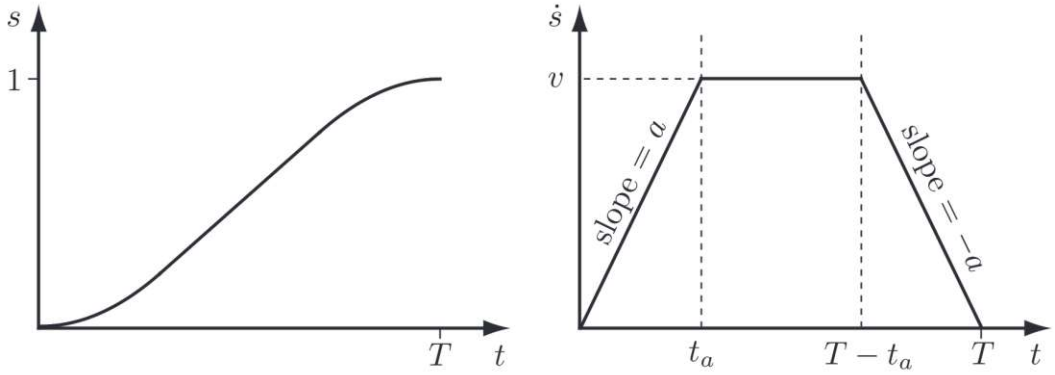


Figure 2.4: Motion profile of a trapezoid time parameterization. To get a smooth start and end phase, a constant slope a is applied. [20]

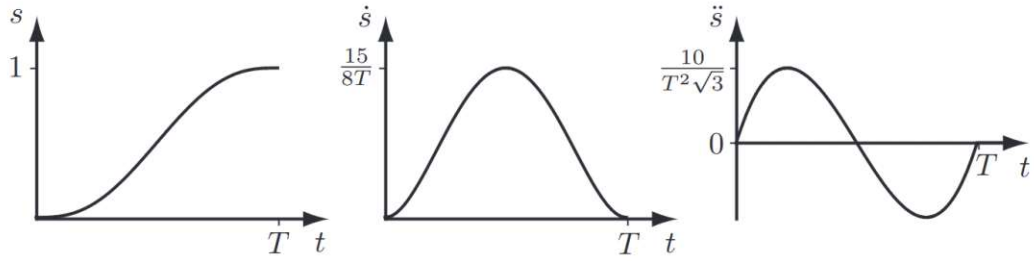


Figure 2.5: Motion profile of a 5th polynomial time parameterization. The maximum velocity and acceleration are limited by the trajectory duration T. [20]

When dealing with more complex paths, time parameterization becomes increasingly difficult, leading to methods such as determining the fastest feasible time scaling that adheres to the robot's actuator limits or optimizing for minimal energy consumption. Moreover, integrating the system's dynamic behavior can introduce varying velocity and acceleration constraints along the path, introducing additional difficulty. [20]

2.4 Iterative Learning Control

Numerous tasks for an industrial robot involve repeating the same operation or motion over and over again. For such tasks, conditions, such as the trajectory and the resulting environmental influence, typically remain consistent with each repetition. ILC leverages this repetitive nature of the task to improve the performance of a system

by correcting these constant undesirable environment influences or model inaccuracies. This improvement is achieved by learning from the observed error information from previous task iterations and refining the input signal of the system on each iteration. This leads to a high control performance in the presence of large uncertainty, in contrast to non-learning controllers which reproduce the same control error in each subsequent iteration. [34]

One of the simplest formulations of an ILC can be expressed as

$$u_{j+1}(t) = u_j(t) + \gamma e_k(t+1), \tag{2.10}$$

where $u_j(t)$ represents the control input at the j-th iteration of the task, γ is the learning gain matrix, and $e_j(t) = y_d(t) - y_j(t)$ is the tracking error, defined as the difference between the desired output $y_d(t)$ and the actual output $y_j(t)$ [35]. A simplified version of the learning process for a singular input value $u_j(t)$ is illustrated in Figure 2.6.

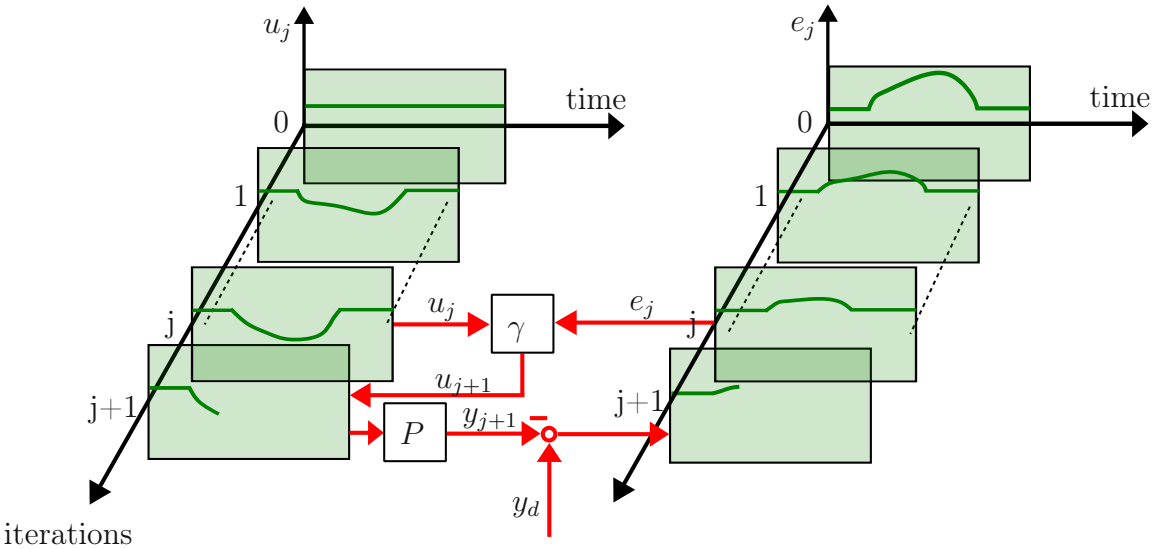


Figure 2.6: Schematic ILC update step for a one-dimensional value. In each iteration j the input value u_j is updated utilizing the error e_j and the learning function γ resulting the new updated input u_{j+1}. The new error e_{j+1} is calculated using the resulting plant output y_{j+1} and the desired output y_d.

2.4.1 The Lifted System Representation

In many applications of ILC, the system might consist of multiple input and output values and is executed over a finite time horizon, where inputs and outputs are recorded at each time step. The lifted system representation aggregates these inputs and outputs into higher-dimensional vectors, also called super-vectors, encapsulating the entire time horizon in a single representation [34].

Consider a linear time-invariant (LTI) discrete-time system

$$\mathbf{x}_j[k+1] = \mathbf{A}\mathbf{x}_j[k] + \mathbf{B}u_j[k] \tag{2.11}$$

$$y_j[k] = \mathbf{C}\mathbf{x}_j[k], \tag{2.12}$$

where $k \in [0, N]$ is the time index, j is the iteration index, $\mathbf{x}_j$ is the state vector, u_j is the control input and y_j the system output.
With $\mathbf{x}_j(0) = \mathbf{x}_0$ for all j, this system is equivalent to

$$y_j(k) = \underbrace{\mathbf{C}(q\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}}_{P(q)} u_j(k) + \underbrace{\mathbf{C}\mathbf{A}^k\mathbf{x}_0}_{d(k)}. \tag{2.13}$$

In this equation, $P(q)$ represents the plant with the forward time shift operator q(e.g., $qz(k) \equiv z(k+1)$ shifts one step forward). The term $d(k)$ represents the free response to the initial condition $\mathbf{x}_0$.
To construct the lifted system representation for a given sample number N, the LTI plant (2.13) is expended as an infinite power series by dividing its denominator into its numerator. Using negative powers of q to represent time delays:

$$P(q) = p_1 q^{-1} + p_2 q^{-2} + p_3 q^{-3} + \dots, \tag{2.14}$$

where $q^{-n}x(k) = x(k-n)$ represents a delay of n time steps. The coefficients $p_k = \mathbf{C}\mathbf{A}^{k-1}\mathbf{B}$ represent the Markov parameters for the given state space description. Markov parameters describe the pulse response functions of a discrete-time linear system to a unit pulse input.[36]
The LTI system (2.13) can now be written as the lifted system

$$\underbrace{\begin{bmatrix} y_j(1) \\ y_j(2) \\ \vdots \\ y_j(N) \end{bmatrix}}_{\mathbf{y}_j} = \underbrace{\begin{bmatrix} p_1 & 0 & \dots & 0 \\ p_2 & p_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ p_N & p_{N-1} & \dots & p_1 \end{bmatrix}}_{\mathbf{P}} \underbrace{\begin{bmatrix} u_j(0) \\ u_j(1) \\ \vdots \\ u_j(N-1) \end{bmatrix}}_{\mathbf{u}_j} + \underbrace{\begin{bmatrix} d(1) \\ d(2) \\ \vdots \\ d(N) \end{bmatrix}}_{\mathbf{d}} \tag{2.15}$$

the error can be also displayed with this representation to

$$\underbrace{\begin{bmatrix} e_j(1) \\ e_j(2) \\ \vdots \\ e_j(N) \end{bmatrix}}_{\mathbf{e}_j} = \underbrace{\begin{bmatrix} y_d(1) \\ y_d(2) \\ \vdots \\ y_d(N) \end{bmatrix}}_{\mathbf{y}_d} - \underbrace{\begin{bmatrix} y_j(1) \\ y_j(2) \\ \vdots \\ y_j(N) \end{bmatrix}}_{\mathbf{y}_j}. \tag{2.16}$$

The lifted form of the ILC learning algorithm (2.10) becomes

$$\underbrace{\begin{bmatrix} u_{j+1}(0) \\ u_{j+1}(1) \\ \vdots \\ u_{j+1}(N-1) \end{bmatrix}}_{\mathbf{u}_{j+1}} = \underbrace{\begin{bmatrix} u_j(0) \\ u_j(1) \\ \vdots \\ u_j(N-1) \end{bmatrix}}_{\mathbf{u}_j} + \underbrace{\begin{bmatrix} l_0 & l_{-1} & \dots & l_{-N-1} \\ l_1 & l_0 & \dots & l_{-N-2} \\ \vdots & \vdots & \ddots & \vdots \\ l_{N-1} & l_{N-2} & \dots & l_0 \end{bmatrix}}_{\mathbf{L}} \underbrace{\begin{bmatrix} e_j(1) \\ e_j(2) \\ \vdots \\ e_j(N) \end{bmatrix}}_{\mathbf{e}_j}, \tag{2.17}$$

where $\mathbf{L}$ represents the learning function.
This lifted representation allows the ILC algorithm to operate on the entire sequence of data, enabling the design of control laws that optimize performance across the whole trajectory rather than at individual time steps.[34]

2.4.2 Designs of Iterative Learning Control

Following the lifted system representation, the learning function $\mathbf{L}$ can be derived through three common methodological approaches: Proportional-Derivative Type (PD-Type), model inversion, or norm-optimal ILC.
The PD-Type ILC combines proportional and derivative components leading to a learning function $\mathbf{L}$ expressed as

$$\mathbf{L}\mathbf{e}_j = \mathbf{K}_p \mathbf{e}_j(t) + \mathbf{K}_d \frac{d\mathbf{e}_j(t)}{dt}. \tag{2.18}$$

Here $\mathbf{e}_j$ is the error of the current iteration j, $\mathbf{K}_p$ is the proportional and $\mathbf{K}_d$ the derivative gain matrices used to tune for a desired performance. This type of ILC is widely used in various applications due to its model-free nature, requiring only effective tuning of two parameter matrices. The stability, robustness, and convergence properties have been extensively analyzed and demonstrated in different scenarios [34].
In order to ensure monotonic convergence and enhance the robustness of the system, it is typically necessary to modify the learning algorithm to include a lowpass Q-filter. The effect of this filter is to effectively disable learning at high frequencies, thereby satisfying the monotonic convergence conditions while simultaneously providing additional robustness and high-frequency noise attenuation. For the PD-Type ILC, a Q-filter is typically comprised of Butterworth, Chebyshev, Gaussian, or FIR type. [34]

Model-Inverse ILC

The model inverse ILC on the other hand uses an inverse of the given system model, leading to the learning function

$$\mathbf{L}\mathbf{e}_j = \mathbf{P}^{-1}\mathbf{e}_j. \tag{2.19}$$

The model inverse ILC approach relies on having an accurate mathematical model of the system, represented by $\mathbf{P}$. With precise model representation, the inverse-based approach can theoretically achieve perfect tracking in a single iteration.
Nevertheless, there exist certain limitations, including the model accuracy, the non-well invertibility of the model due to non-minimum phase characteristics, input-output

dimension mismatch, and singularities in the system model [37].

Norm-Optimal ILC

While PD-ILC offers simplicity and model-free implementation, and model inverse ILC faces practical limitations, norm-optimal ILC functions as an intermediate approach. It formulates the learning as an optimization problem that minimizes a cost function, typically described as

$$J = \hat{\mathbf{e}}_{j+1}^T \mathbf{W}_e \hat{\mathbf{e}}_{j+1} + \mathbf{u}_{j+1}^T \mathbf{W}_u \mathbf{u}_{j+1} + \boldsymbol{\delta}\mathbf{u}_{j+1}^T \mathbf{W}_{\boldsymbol{\delta}u} \boldsymbol{\delta}\mathbf{u}_{j+1}, \tag{2.20}$$

where $\boldsymbol{\delta}\mathbf{u}_{j+1} = \mathbf{u}_{j+1} - \mathbf{u}_j$ and $\hat{\mathbf{e}}_{j+1} = \mathbf{e}_j - \hat{\mathbf{P}}\boldsymbol{\delta}\mathbf{u}_{j+1}$ is the predicted next iterations output error. $\mathbf{W}_e$,$\mathbf{W}_u$ and $\mathbf{W}_{\delta u}$ are the positive semi-definite weighing matrix. [38]
From this the learning function $\mathbf{L}$ can derived as

$$\mathbf{L} = (\hat{\mathbf{P}}^T \mathbf{W}_e \hat{\mathbf{P}} + \mathbf{W}_{\delta u})^{-1} (\hat{\mathbf{P}}^T \mathbf{W}_e). \tag{2.21}$$

The norm-optimal approach provides a robust framework that reduces the sensitivity to modeling uncertainties compared to inversion-based ILC while offering systematic tuning through the weighting matrices.

2.4.3 ILC in Industrial Robot Applications

In the context of industrial robots, there are several approaches to increase their path-tracking capability by using an ILC approach to mitigate recurring errors. These methods are typically implemented as augmentations to existing control architectures, such as feed-forward and feedback control systems, or employed in robot kinematic calibration procedures. The error feedback for these systems is obtained through the internal robot joint encoders in conjunction with additional external sensors such as torque sensors, lasers, or camera systems. [39–41]
As in other areas of robotics, camera systems are available both as eye-in-hand and externally mounted applications. One vision-based approach utilizes marker detection for error estimation, where a monocular eye-in-hand camera tracks visual markers to infer end-effector deviations from a demonstrated trajectory [42]. Applications in the field of visual sensing and visual odometry are not currently available [43].
To handle the inherent nonlinear dynamics of industrial robots, one approach treats nonlinearities as perturbations to a linearized system model. Under feedback control, the robot's trajectory typically remains within a neighborhood of the reference path, allowing the nonlinearities to be evaluated along this reference trajectory. This results in an additive Linear Time-Varying (LTV) system where the perturbations remain constant across iterations [44].

2.5 Image Formation

Image formation is the concept in computer vision to capture and represent a 3D scene onto a Two-dimensional (2D) image plane using a camera system. This procedure consists of transforming world coordinates into camera coordinates through rigid transformations $\mathbf{T}_{\mathcal{W}}^{\mathcal{C}}$, followed by projecting these camera coordinates onto pixel coordinates, as depicted in Figure 2.7.

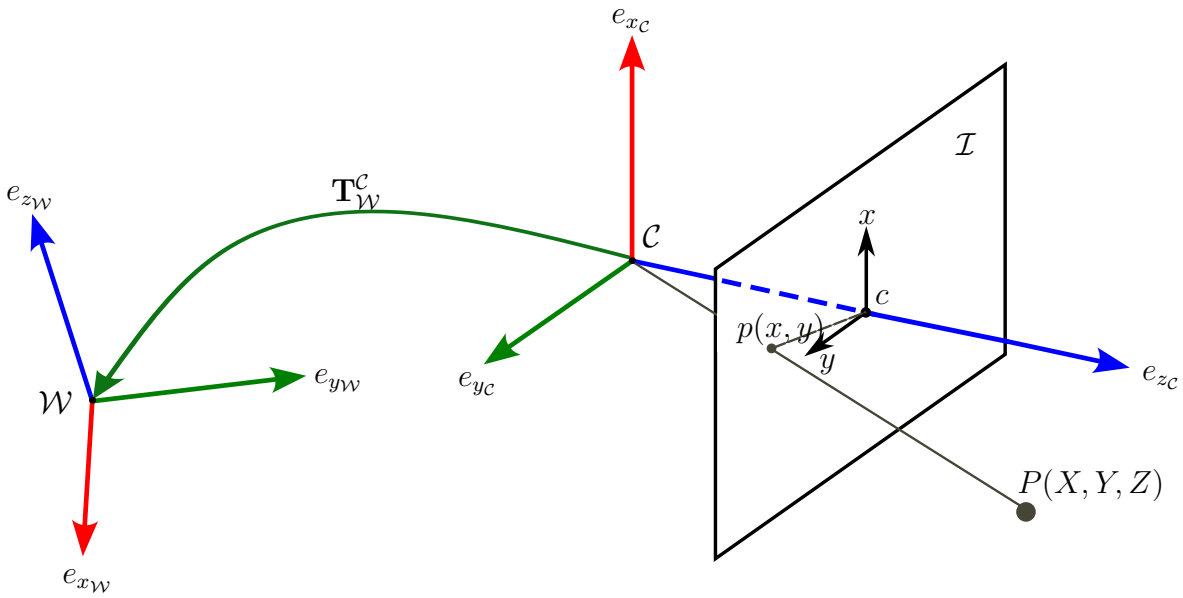


Figure 2.7: Illustration of a complete camera model. $\mathcal{W}$ is the 3D world coordinate system and $\mathcal{C}$ the camera coordinate system. $\mathcal{I}$ represents the image frame of the camera with its own 2D coordinate system c. The 3D-point P is projected onto $\mathcal{I}$ which maps to the 2D-point, resulting in p.

2.5.1 Pinhole Camera Model

The pinhole camera model offers a fundamental method for projecting 3D points onto the image plane, simplifying the complexities found in real optical systems, and providing a theoretical foundation in camera optics [45]. It condenses the camera aperture to a singular point where light rays converge, termed the pinhole, enabling a simplified projection of an image onto a plane.
The resulting projection leads to the mathematical description of the relationship between points in the 3D space and its 2D image frame position (cf. Equation (2.22) and (2.23)). The focal length f is used in these equations to determine the projection of the 3D point P onto the image plane, specifying its x and y coordinates. Figure 2.8 illustrates a 2D version of a pinhole camera model.

$$y = f \cdot \frac{P_y}{P_z} \tag{2.22}$$

$$x = f \cdot \frac{P_x}{P_z} \tag{2.23}$$

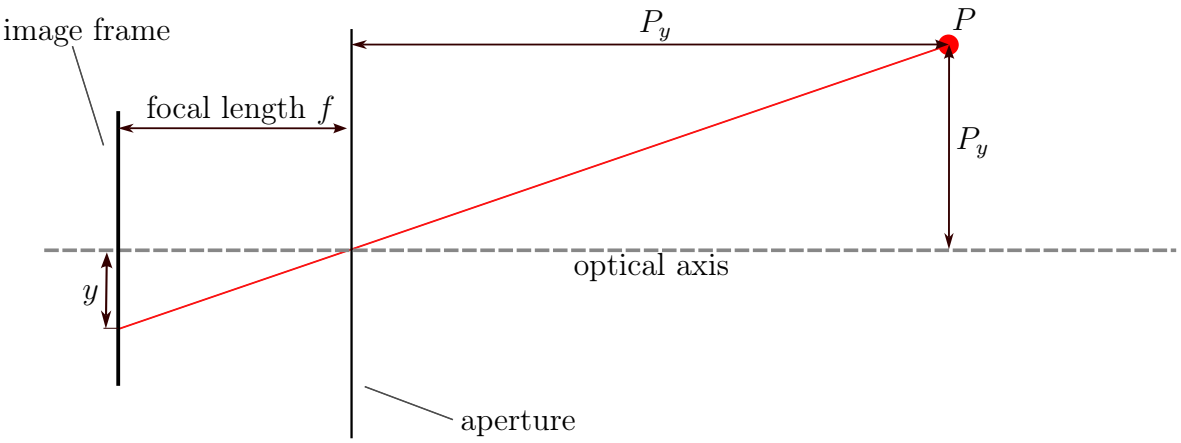


Figure 2.8: Working principal of a 2D pinhole camera model. The 3D point P is projected onto the image plane based on the focal length f.

Although the pinhole camera model is useful for a basic overview, it has significant limitations in practice. First, it does not account for radial and tangential distortions introduced by lenses, which can significantly affect the accuracy of image representation [46]. In addition, the continuous nature is captured as an array of discrete pixels, specified by the image sensor. This leads to a potential loss of information, which is only present in subpixel resolution.

2.5.2 Camera Parameters

The transformation of the coordinate systems and projection of images onto the camera sensor can be further described with camera parameters. Extrinsic parameters define the rigid transformation, and intrinsic parameters describe the projection from the 3D scene point to the 2D pixel values. The extrinsic camera parameters are represented by a homogeneous transformation matrix, described in detail in Section 2.2.1.

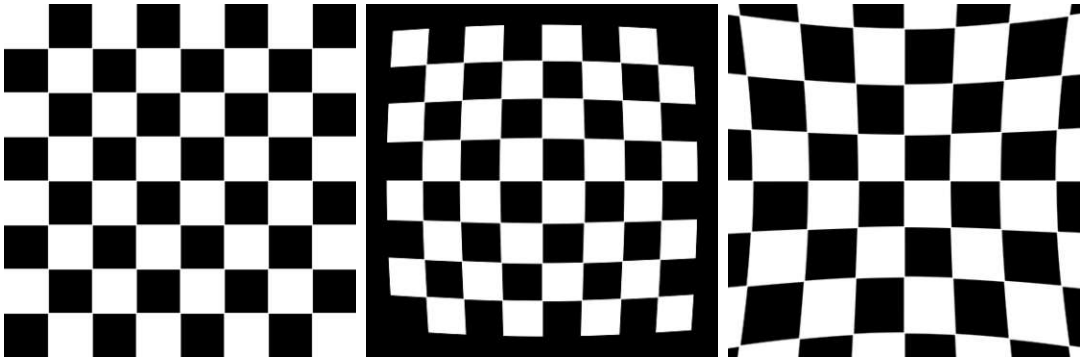
$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{2.24}$$

The intrinsic parameters can be expressed as the matrix $\mathbf{K}$:

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.25}$$

Here, f_x and f_y represent the focal lengths along the x and y axes of the image plane, c_x and c_y are the coordinates of the principal, typically the image center, and the skew coefficient s.
With these parameters, the pinhole camera model can be elevated to:

$$r \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{2.26}$$



(a) Uniform Chessboard (b) Barrel Distortion (c) Pincushion Distortion

Figure 2.9: Illustration of different types of image distortions using a chessboard pattern. (a) The Uniform Chessboard represents a standard grid with no distortion. (b) The Barrel Distortion demonstrates the outward bulging effect, typical of wide-angle lenses, where straight lines appear to curve away from the center. (c) The Pincushion Distortion, where straight lines curve towards the center.

2.5.3 Camera Calibration

Both the intrinsic camera parameters and the distortion of the lens can be determined by calibrating the camera to be used. The most common method for camera calibration involves a planar calibration target [47, 48]. This planar calibration target is used to provide a known geometric structure that is easily detected and measured in images captured by the camera. In Figure 2.9, a chessboard pattern is illustrated with an illustrated barrel and pincushion distortion and can be used for calibration.
By observing the apparent distortion in these structured images, the calibration process can determine the camera's intrinsic parameters and the distortion.

2.6 Depth Estimation

As previously stated, a camera's ability to capture a 2D projection of a 3D world is inherently limited. However, in numerous applications within the fields of robotics and computer vision, the precise position of objects in three dimensions is essential. Consequently, in addition to the captured image, the depth information from the camera's viewpoint is crucial for comprehending the entire scene.
Numerous methods can be used to extract depth information from a scene, either by utilizing or complementing the spatial data obtained in the image.
One reasonably intuitive option is the use of external sensors such as Light Detection and Ranging (LiDAR), which is based on the Time of Flight method and determines the distance by measuring the time between a transmitted light signal and its reflected signal. However, LiDAR systems can be expensive and cumbersome and may struggle in certain environments, such as those with highly reflective or absorptive surfaces. [49]
As an alternative, depth in a scene can also be estimated without additional sensors, utilizing images taken by a single camera. This process can be divided into Single Image

Depth Estimation (SIDE) and multiple-view geometry techniques.
For SIDE, machine learning methods, specifically convolutional neural networks, are used almost exclusively [50]. Both supervised and unsupervised models are used, often divided into relative and absolute depth, as well as indoor and outdoor environments.
In multiple-view geometry, depth information necessitates at least two different scene views. Techniques such as stereo vision or Structure from Motion (SfM), which rely on triangulation principles, are commonly employed [45].
Stereo vision captures two or more images using cameras positioned at a known distance apart. By examining the differences between corresponding points in these images, depth information can be estimated. One significant challenge is the need for precise camera calibration to ensure accurate depth estimation. In addition, stereo vision systems can struggle with textureless or repetitive surfaces, making it difficult to find the corresponding points. Occlusions, where one object blocks the view of another, can also pose problems by creating gaps in the depth map. [51]
SfM uses an image series of a static scene taken from different viewpoints. These images are used to create a 3D reconstruction of the scene, most popularly using an incremental approach.
The incremental SfM method starts by reconstructing a 3D scene using a set of initial images. This is achieved by the detection and matching of features in these images (cf. Section 2.7.1). Afterward, more images are progressively added, refining the 3D structure and camera parameters with each addition. [52]

2.6.1 Triangulation

Triangulation is a fundamental technique used to reconstruct the 3D coordinates of a point by intersecting the lines of sight from at least two distinct camera views. This method is deeply rooted in epipolar geometry, which provides a geometric framework that simplifies the search for corresponding points between two images.

Epipolar Geometry

Using the epipolar geometry shown in Figure 2.10, the relationship between the two projected points x and x' can be simplified. If the position of x in the frame $\mathcal{I}$ and the camera origin C' are known, x' can be found only on the epipolar line l'. Thus, considerably reducing the search area.

The Fundamental Matrix

The fundamental matrix $\mathbf{F}$, a 3x3 matrix, encapsulates the algebraic representation of epipolar geometry [45]. It ensures that for any corresponding point pair $x \leftrightarrow x'$ across two images, the following condition holds:

$$(x')^T \mathbf{F} x = 0. \tag{2.27}$$

This is also called the epipolar constraint.
While $\mathbf{F}$ is applicable in scenarios with uncalibrated cameras and includes both intrinsic and extrinsic parameters, $\mathbf{E}$ is used when cameras are calibrated, containing only

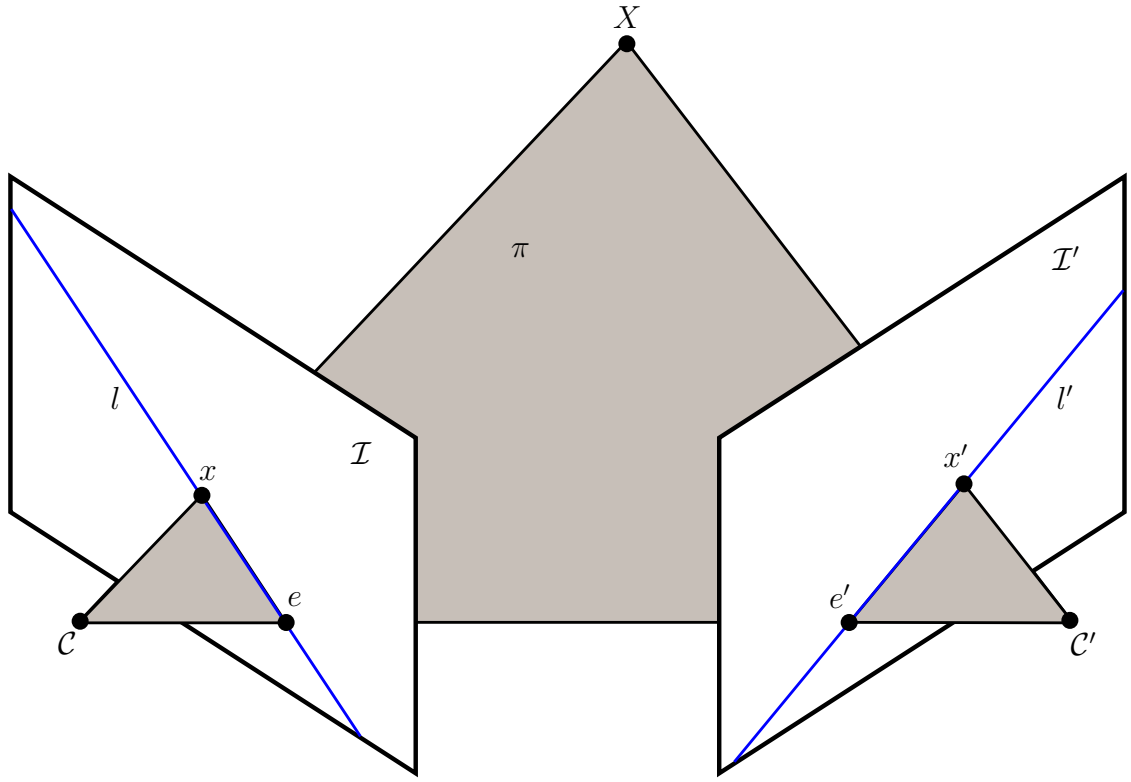


Figure 2.10: Epipolar geometry: The 3D point X is projected onto the image frames $\mathcal{I}$ and $\mathcal{I}'$ using the pinhole camera model as described in Section 2.5.1. The cameras are indicated by their origin $\mathcal{C}$ and $\mathcal{C}'$. The projections of the respective other camera origin form the epipoles e and e'. The lines between x and e then form the epipolar lines l and l'. The projected points, together with the epipoles, span the epipolar plane π.

the extrinsic parameters rotation and translation. With the known intrinsic camera parameter $\mathbf{K}$, the relationship between $\mathbf{F}$ and $\mathbf{E}$ is:

$$\mathbf{E} = (\mathbf{K})^T \mathbf{F} \mathbf{K}. \tag{2.28}$$

The essential matrix can also be decomposed using SVD to recover the rotation $\mathbf{R}$ and translation $\mathbf{t}$, based on the recomposition theorem in [53]

$$\mathbf{E} = \mathbf{U}\, diag(1, 1, 0)\mathbf{V}^T, \tag{2.29}$$

where $det(\mathbf{U}) > 0$ and $det(\mathbf{V}) > 0$. The translation vector $\mathbf{t}$ is obtained by decomposing $\mathbf{U}$ into its scalar matrix elements u_{ij}

$$\mathbf{t} = [u_{13}, u_{23}, u_{33}]^T \tag{2.30}$$

and the rotation matrix $\mathbf{R}$ is chosen as

$$\mathbf{R}_a = \mathbf{U}\mathbf{D}\mathbf{V}^T \quad \text{or} \quad \mathbf{R}_b = \mathbf{U}\mathbf{D}^T\mathbf{V}^T, \tag{2.31}$$

where $\mathbf{D}$ is a matrix defined as:

$$\mathbf{D} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.32}$$

These decompositions result in four potential configurations for the camera's pose relative to its initial position

$$\begin{aligned} \mathbf{P}_A &= [\mathbf{R}_a|\mathbf{t}] \quad \mathbf{P}_B = [\mathbf{R}_a|-\mathbf{t}] \\ \mathbf{P}_C &= [\mathbf{R}_b|\mathbf{t}] \quad \mathbf{P}_D = [\mathbf{R}_b|-\mathbf{t}]. \end{aligned} \tag{2.33}$$

To resolve these ambiguities, the chirality condition is applied, which ensures that triangulated points are in front of both cameras, being physically feasible.

Algorithms

The triangulation problem can be summarized as finding the intersection of the two projection lines. In the presence of noise, these projection rays are not guaranteed to cross, introducing complexity into the system [54]. A widely used approach to account for noise involves optimizing a cost function that minimizes the re-projection error to determine the best image points $\hat{x}$ and $\hat{x}'$. The goal is to find the points $\hat{x}$ and $\hat{x}'$ which minimize Equation (2.34), where $d(\square, \square)$ indicates the Euclidean distance and is subject to the epipolar constraint.

$$d(x, \hat{x})^2 + d(x', \hat{x}')^2 \tag{2.34}$$

This minimization process can be achieved by utilizing iterative approaches like iterative least square minimization methods such as the Levenberg–Marquardt algorithm (LM) [55].
Alternatively, it can also be done with a non-iterative algorithm like the Polynomial method, which transforms the problem into a sixth-order polynomial. The roots of this polynomial then represent potential solutions to the minimum of the cost function. [54]

2.7 Visual Odometry

Visual Odometry (VO) is the process of estimating the movement of an agent equipped with one or more cameras [56]. This movement, known as egomotion, refers to the 3D motion of a camera within an environment, which includes tracking the camera's changes in position and orientation over time [57]. The methodologies for VO are diverse, utilizing various types of equipment to meet specific application requirements. These methodologies can be categorized into three main types: monocular, stereo, and visual-inertial odometry.
Monocular odometry relies on a single camera mounted on the agent to estimate motion. This method is advantageous due to its simplicity and lower hardware requirements but it often faces challenges related to scale ambiguity and depth estimation. [56]

Stereo visual odometry uses two cameras to capture the scene. These cameras are placed at a known distance apart, allowing an accurate depth perception by triangulation [58]. Visual-inertial odometry combines visual data from a camera with inertial measurements from Inertial Measurement Units (IMUs). This fusion of data sources improves the accuracy and robustness of motion estimation, especially in scenarios with fast movements or low-texture environments. [59–61]
These hardware setups can also be extended with additional computer vision methods. For example, deep learning is used in combination with monocular VO to obtain additional depth data from the scene [62].
In this thesis, monocular visual odometry is highlighted due to its minimal hardware requirements as well as the special application case, which makes it relatively easy to generate depth data with just one camera (cf. Section 3.2.1). Therefore, the next section examines the details and methodologies related to monocular VO.

2.7.1 Monocular Visual Odometry

Monocular VO only uses image sequences from a single camera to estimate motion. To accomplish this estimation, there are two main methods: feature-based and appearance-based methods. Feature-based methods rely on detecting and matching distinct points or features in images, while appearance-based methods utilize the overall intensity patterns or textures in the images to estimate motion. However, because feature-based methods are more accurate than appearance-based ones, most implementations of VO are based on them. [63]
Consequently, this thesis will also focus on feature-based methods.
Thereby, features describe regions within an image, such as edges, corners, or lines,

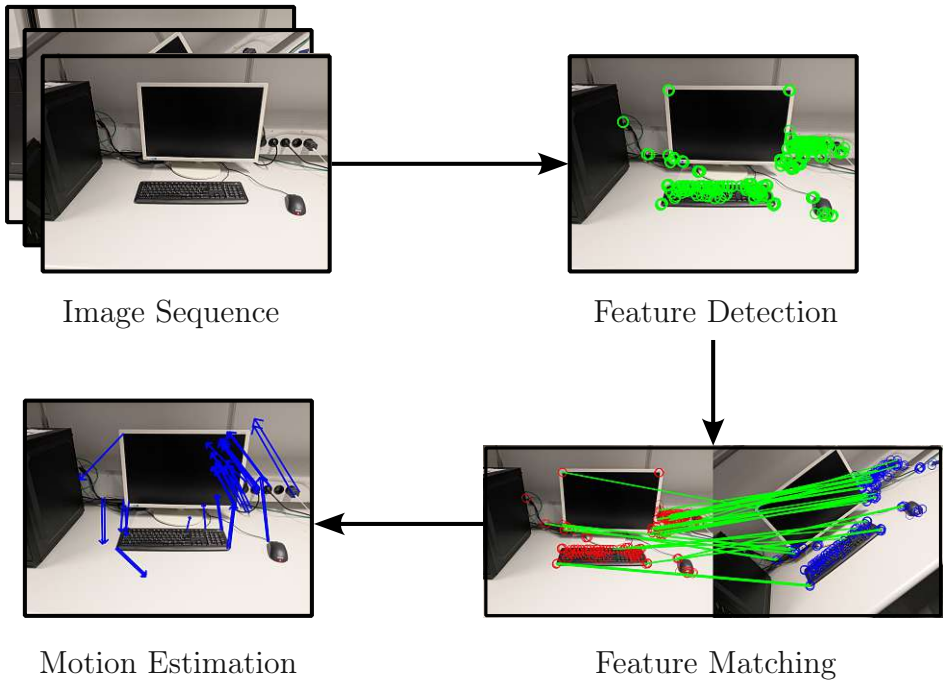


Figure 2.11: Illustrative flowchart of a typical feature-based monocular VO process. [56]

generating a pattern that is distinguishable from its immediate neighborhood in terms of intensity, color, and texture [56]. These features are used to detect the motion of the agent in the environment. The fundamental workflow of feature-based VO, which is subsequently further described, is illustrated in Figure 2.11.

Feature Detection

Because VO needs the reliable detection and matching of distinctive points across consecutive frames, feature detectors like Scale-Invariant Feature Transform (SIFT) [64], SURF [65], ORB [66] or FAST [67] are used. A reliable feature detector is crucial for effective and accurate estimation of the camera motion and, therefore, has to exhibit several key properties [56]. These include:

- **Localization Accuracy**: Precise detection of feature locations and scales within an image is essential, as it serves as the foundation for accurate motion estimation.
- **Repeatability**: A substantial number of the same features should consistently be detected across the sequence of images. This is necessary to allow the matching of features in the image sequence.
- **Robustness**: The detector must be resilient against various image degradations such as noise, compression artifacts, and blur.
- **Distinctiveness**: Features should be sufficiently unique to allow for precise matching across different images.
- **Invariance**: The detector should be invariant to changes in lighting conditions and geometric transformations such as rotation, scaling, and perspective distortion, as this ensures reliable feature detection and tracking under various environmental conditions and viewpoints.

Feature Matching

After the features of each image are identified, a matching procedure is carried out within the sequence of images to spatially relate the images. One approach is to track the features using local search techniques like Kanade–Lucas–Tomasi (KLT) feature tracker [68]. This feature tracker uses the concept of optical flow throughout the image sequence to track feature points. Optical flow assumes that the intensity of features remains constant between the image sequence, allowing the algorithm to compute the displacement of each feature by minimizing the difference in intensity. This approach is common for small changes in the view of the images and might introduce a motion drift over the sequence. [56]
Another approach is to detect the feature points in all images and match the descriptors to each other to find the identical scene points.
Descriptors are numerical representations that uniquely identify and encapsulate the information of a feature, facilitating reliable matching across different images. These descriptors are computed from the local image patches around each feature point, using algorithms designed to maximize distinctiveness and invariance to common image

transformations, like the SIFT descriptor [64].
The identified and located features in both images are subsequently matched to each other using algorithms such as Brute Force or Fast Library for Approximate Nearest Neighbors (FLANN) [69].
In Brute Force matching, each descriptor in one image is compared to each descriptor in the other image using distance functions such as Euclidean, Manhattan, or Hamming distance to measure similarity, selecting the closest matches based on these calculated distances.
FLANN is a library designed to speed up the process of matching high-dimensional descriptors by automatically selecting the most efficient approximate nearest neighbor algorithm and parameters for a given dataset. It supports various methods, such as multiple randomized k-d trees and hierarchical k-means trees, to significantly reduce query time while maintaining high accuracy. [69]
To increase the robustness of the matching, bad matches introduced by outliers are rejected afterward by the use of Random Sample Consensus (RANSAC) [70] or more specific approaches like the Lowe Ratio Test [71].

Motion Estimation

Once sufficient features have been detected and matched in the image sequence, a motion estimation process is required to determine the spatial transformation of the agent between the corresponding images. The basic concept to determine this transformation $\mathbf{T}_{\mathcal{E}_k}^{\mathcal{E}_{k-1}}$ over two successive images is shown in Figure 2.12.
Leveraging the epipolar constraints, the essential matrix $\mathbf{E}$ (cf. Section 2.6.1) is estimated, from which the rotation $\mathbf{R}$ and translation $\mathbf{t}$ are recovered. This estimation process commonly employs methods such as the normalized eight-point algorithm [72] or the five-point algorithm [53]. These algorithms require at least eight or five pairs of corresponding points from two distinct images, respectively, leveraging the epipolar constraints to formulate polynomial equations. These equations are typically solved using numerical methods such as SVD or Eigenvalue Decomposition. After the estimation of the essential matrix, it can be decomposed, as described in Section 2.6.1.
To improve accuracy in real-world applications affected by noise and outliers from wrong feature point matches, these methods are typically combined with the RANSAC algorithm. RANSAC iteratively applies them to random subsets of data and selects the essential matrix that best fits most of the sampled data.
In addition to these algebraic solutions, there are also iterative methods. These transform the pose-solving problem into a non-linear least square optimization problem that needs to be solved [73].

2.7.2 Evaluation Metrics

The evaluation of an estimated trajectory $\hat{\mathbf{x}}$ and its ground truth $\mathbf{x}^{gt}$ is not straightforward, as it is only possible to compare the individual waypoints at given times of the trajectories. To calculate the error between these trajectories, two common error metrics are the Absolute Trajectory Error (ATE) and the Relative Error (RE). [74]

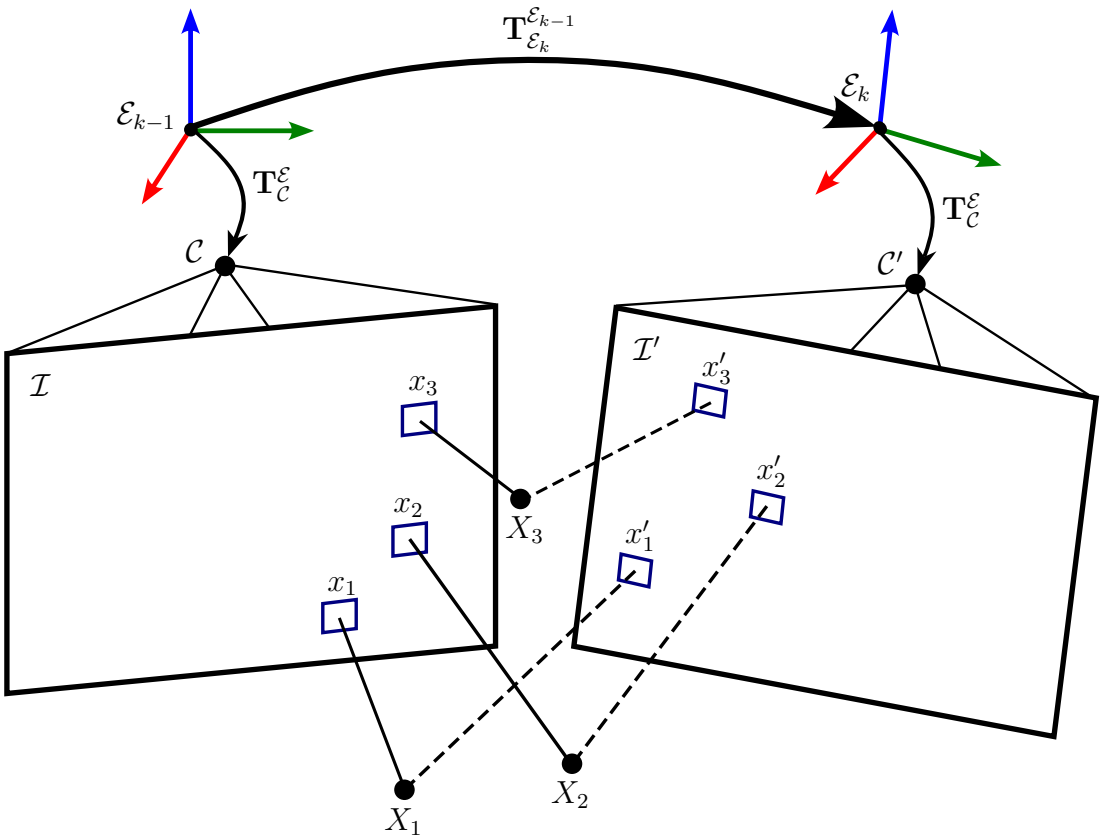


Figure 2.12: Transformation $\mathbf{T}_{\mathcal{E}_k}^{\mathcal{E}_{k-1}}$ between consecutive agent poses $\mathcal{E}_k$ and $\mathcal{E}_{k-1}$, derived from corresponding 3D points X_i observed from camera positions $\mathcal{C}$ and $\mathcal{C}'$. The transformation is computed using image projections x_i and x'_i through motion estimation. The camera-to-agent transformations $\mathbf{T}_{\mathcal{C}}^{\mathcal{E}}$ and $\mathbf{T}_{\mathcal{C}'}^{\mathcal{E}}$, which link the agent to the camera origins, remain constant across k

Absolute Trajectory Error (ATE)

For a single waypoint of a trajectory the error between $\hat{\mathbf{x}}_i$ and $\mathbf{x}_i^{gt}$ can be parameterized as

$$\Delta\mathbf{x}_i = \{\Delta\mathbf{R}_i, \Delta\mathbf{p}_i, \Delta\mathbf{v}_i\}. \tag{2.35}$$

$\Delta\mathbf{R}_i, \Delta\mathbf{p}_i$ and $\Delta\mathbf{v}_i$ represent here the rotation error, the position error, and the velocity error of the end-effector pose at the waypoint i and can be calculated with

$$\Delta\mathbf{R}_i = \mathbf{R}_i(\hat{\mathbf{R}}_i)^T \tag{2.36}$$

$$\Delta\mathbf{p}_i = \mathbf{p}_i - \Delta\mathbf{R}_i\hat{\mathbf{p}}_i \tag{2.37}$$

$$\Delta\mathbf{v}_i = \mathbf{v}_i - \Delta\mathbf{R}_i\hat{\mathbf{v}}_i. \tag{2.38}$$

To evaluate the complete trajectory, the root-mean-square error is calculated across its waypoints. This approach leads to the definitions of the rotation, position, and velocity absolute trajectory errors (ATE_{rot}, ATE_{pos}, and ATE_{vel}) as shown in Equation (2.39),

Equation (2.40), and Equation (2.41).

$$\mathrm{ATE}_{\mathrm{rot}} = (\frac{1}{N}\sum_{i=0}^{N-1} ||\angle(\Delta\mathbf{R}_i)||^2)^{\frac{1}{2}} \tag{2.39}$$

$$\mathrm{ATE}_{\mathrm{pos}} = (\frac{1}{N}\sum_{i=0}^{N-1} ||\Delta\mathbf{p}_i||^2)^{\frac{1}{2}} \tag{2.40}$$

$$\mathrm{ATE}_{\mathrm{vel}} = (\frac{1}{N}\sum_{i=0}^{N-1} ||\Delta\mathbf{v}_i||^2)^{\frac{1}{2}} \tag{2.41}$$

Here, $\angle(\square)$ describes the conversion of the rotation matrix $\mathbf{R}$ to an angle axis repre-

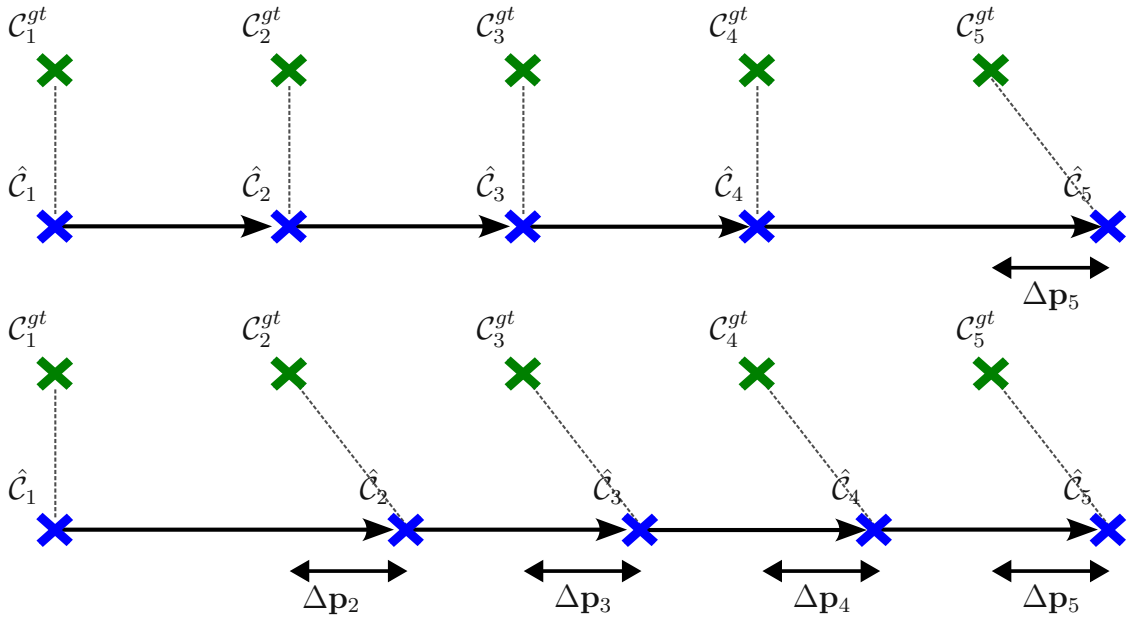


Figure 2.13: Simplified example for the timing problem of an ATE trajectory calculation. A robot end-effector executes the same trajectory twice, moving from $\mathcal{C}_1$ to $\mathcal{C}_5$. In the upper illustrated trajectory execution, a position error $\Delta\mathbf{p}_5$ occurs at the last waypoint, resulting in a low global error. In the lower trajectory, the same error occurs earlier at $\hat{\mathcal{C}}_2$, propagating through the subsequent waypoints ($\Delta\mathbf{p}_2$, $\Delta\mathbf{p}_3$, $\Delta\mathbf{p}_4$, and $\Delta\mathbf{p}_5$) and causing a larger cumulative global error.

sentation via Rodrigues [75].
The ATE now describes the error of the whole trajectory in a single metric scalar number for rotation, position, and velocity estimation. However, ATE is sensitive to errors occuring during the trajectory, illustrated in Figure 2.13 [76].
Therefore, as an alternative calculation, the RE is additionally used for a more informative evaluation.

Relative Error (RE)

The RE measures the accuracy of trajectory estimation by comparing pairs of states at different points along the path, rather than using absolute positions. By calculating the rotational and positional differences between the estimated and actual states for different trajectory segments, RE provides information on both local consistency and long-term accuracy. While this metric is valuable for systems without a global reference frame, its complex computational method and multidimensional error representation make it less intuitive than single-value metrics like ATE. [74]

2.8 Robot Operating System (ROS)

The ROS is a flexible framework designed to aid in the development of robotic setups. It provides comprehensive hardware abstraction and low-level device control and facilitates message-passing between processes through a robust package management system [77, 78].
Additionally, ROS supports the seamless integration and management of complex robotic components through an efficient communication framework comprising nodes, topics, services, and messages. Nodes are processes that perform computations and communicate by publishing or subscribing to data streams called topics. Messages define the data types used in this data stream.
Services in ROS, are an alternative interaction between nodes, providing a synchronous, request/response interaction paradigm. This architecture allows for a robust and flexible way to manage the complexities of robotic programming, ensuring that systems can be efficiently controlled and data flows smoothly between components. The whole communication is managed by the ROS-Master, which manages interactions between nodes and maintains reliable networking across the system, as detailed in Figure 2.14. This networked approach is highly advantageous in robotic development, as it allows for the effortless integration of subsystems into a larger, pre-existing system. The modular approach makes debugging easier, as nodes in development can run concurrently with those that have already been thoroughly debugged.
Furthermore, ROS also offers valuable tools like RVIZ, a service that visualizes different types of messages including images, poses, or point clouds [79]. To handle information storage in a running setup, ROSBAGS can be utilized to store serialized message data in files, including the precise timestamps of when each topic was published [80]. Their ability to record multiple topics simultaneously and with accurate timing makes ROSBAGS an excellent resource for capturing and debugging robot operation details during runtime.
Given the complexity of robotics and the numerous challenging subtasks involved, the thin structure of ROS allows developers to easily integrate existing software libraries like OpenCV [81], Ceres Solver [82], or MoveIt![83] as well as drivers for different hardware like the camera_aravis package openly available in GitHub [1]. This opens up the possibility of utilizing established algorithms and tools, avoiding the need to develop these components from scratch.

[1] https://github.com/FraunhoferIOSB/camera_aravis

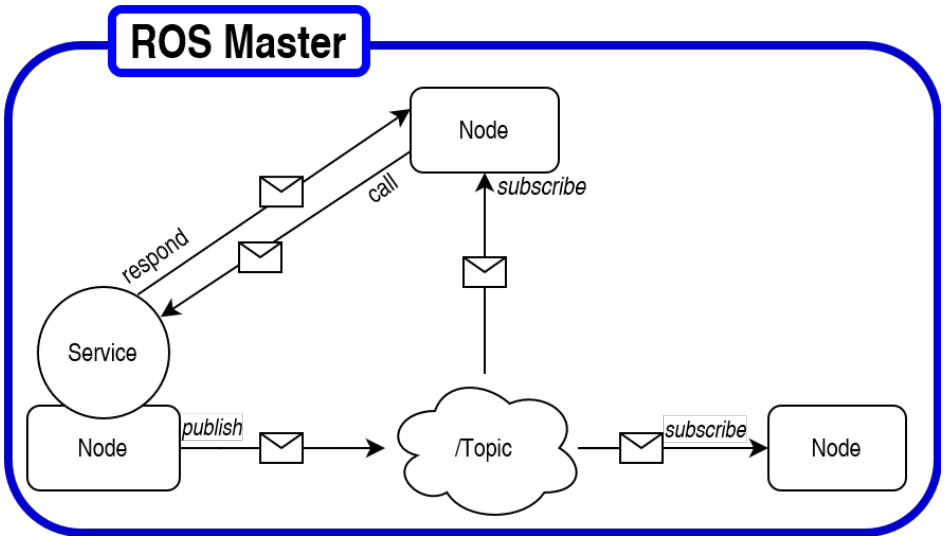


Figure 2.14: Depiction of a basic ROS program structure, featuring three nodes and a single topic, with one node also functioning as a service server. The topic is displayed with a cloud symbol, where the nodes publish and subscribe from. All nodes and topics are registered with the ROS Master, which manages them.

2.8.1 MoveIt! Motion Planning Framework

MoveIt! is a software framework for motion planning and mobile manipulation within the ROS ecosystem [83]. One of the key features of MoveIt! is its ability to maintain a clear distinction between its core functionalities and the robotic framework-dependent aspects, such as component communication, which are typically managed via the core ROS build and messaging systems. This separation allows developers to focus on high-level motion planning tasks without needing to handle low-level communication intricacies.
Furthermore, MoveIt! employs a plugin-based architecture for essential functions including motion planning with Open Motion Planning Library (OMPL) [84], collision detection using the Fast Collision Library [85], and kinematics handled by the OROCOS Kinematics and Dynamics Library for both forward and inverse kinematics [86]. This modular design not only simplifies the comparison of different algorithms by enabling easy swapping and testing of various plugins but also eliminates the need for developers to implement these complex algorithms from scratch.

CHAPTER 3

Design and Implementation of a vision-based Iterative Learning Control System

The proposed system aims to identify and mitigate end-effector pose errors using an eye-in-hand camera system and Visual Odometry (VO) for relative pose estimation. The joint errors calculated from these estimations are then iteratively minimized through Iterative Learning Control (ILC). The process begins by capturing a ground truth trajectory under optimal conditions, such as slow movements, light payloads, or hand-guided teaching, to establish baseline measurements. Subsequent iterations are performed under actual operating conditions, with continuous trajectory refinement, as visualized in Figure 3.1.

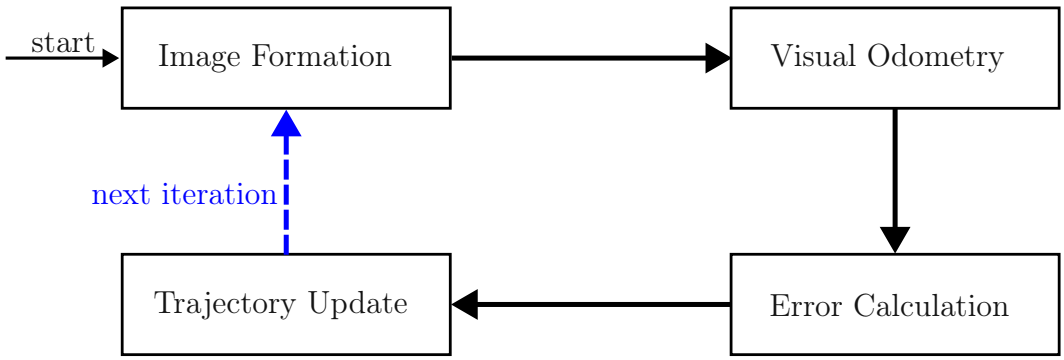


Figure 3.1: Single iteration of the ILC setup, updating all waypoints of a trajectory.

3.1 Hardware Setup

For the vision-based ILC to detect pose errors and employ the implemented subsequent compensation, the hardware setup must be carefully designed. At its core, the system employs a six-axis rigid industrial robot equipped with a camera mounted on its end-effector, also known as an eye-in-hand setup. This end-effector mounting configuration

allows the camera to track unwanted changes in motion that arise from both kinematic and non-kinematic errors.
To ensure the system can detect even slight deviations in movement, the camera's properties must be precisely calibrated to align with the trajectory's specific requirements. Additionally, the communication between the robot, the camera, the vision algorithm hardware setup, and its control is also crucial to ensure the needed constant iterative nature of the ILC. The communication flow between the system components is depicted in Figure 3.2, illustrating the interaction between Robot Operating System (ROS), the robot system, and the camera for trajectory execution and image acquisition.

Figure 3.2: System architecture depicting the communication flow between hardware components (camera and robot) and ROS. The robot executes the given trajectory and sends continuous joint positions back to ROS. These positions are to the desired values and subsequently trigger the camera to capture and send an image.

3.1.1 Robot System

When selecting a robot system, the repeatability of the axis positioning movements plays an important role. Since the algorithm updates joint angles iteratively, the robot must reliably achieve commanded positions to prevent non-compensatable errors in the control loop. Therefore, robots with internal Proportional-Integral-Derivative (PID) control can be used to compensate for the error deviation within the joint.
Additionally, compensation for both translational and rotational errors requires full spatial maneuverability. Therefore, at least a 6-axis robot configuration is necessary to provide the required six degrees of freedom for comprehensive error correction. While 7-axis robots offer additional flexibility, they introduce control complexity due to redundant configurations for the same end-effector pose, complicating inverse kinematics and motion planning.

3.1.2 Camera System

The camera system is an essential component of the thesis, acting as the sole sensor for the ILC. The role of the camera during the ILC is to capture an image of the environment at each waypoint of the trajectory, which will later be used in the motion error estimation (cf. Section 3.2). As the camera progresses along the trajectory, it detects numerous environmental features. To ensure these features are captured accurately, it is necessary to account for the minimal detectable movement of a projected

environment feature.
This minimum motion detection depends on the camera's resolution, Field of View (FoV), and frame rate. However, two of these properties, the frame rate, and the resolution, are constrained by the available bandwidth, which is determined by the interface used. A higher bandwidth allows for an increased trajectory sampling rate and enables the execution of faster trajectories.
Table 3.1 compares data transfer rates for various commonly used interfaces.
Using the potential maximum data transmission as a reference, several limitations

Table 3.1: Comparison of data transfer rates for different interfaces

Interface	**Data Transfer Rate (Gbit/s)**
USB 2.0	0.480
USB 3.1 Gen 2	10
USB 3.1 Gen 1 (USB 3.0)	5
Gigabit Ethernet	1
10-Gigabit Ethernet	10
PCI Express 2.0 (x1 lane)	8

of the vision system are analyzed, including resolution, FoV, frame rate, and motion blur. These characteristics are interdependent, complicating the selection of an optimal configuration.

Pixel Resolution

The pixel resolution of an image sensor fundamentally limits the accuracy with which the pose error of the end-effector can be detected. It determines the smallest translational and angular changes that can be observed and later used in the ILC.
To reliably detect any translational movement of the end-effector, the smallest observable environment shift in the image is at least one pixel, represented as $\Delta u = 1$ or $\Delta v = 1$. The perspective projection model can express the minimal detectable motion, relating pixel displacements to real-world translations. This is described by:

$$\Delta x_{\min} = \frac{p_z}{f}\Delta u, \quad \Delta y_{\min} = \frac{p_z}{f}\Delta v, \tag{3.1}$$

where f denotes the focal length, p_z the target point distance, and $\Delta x_{\min}$, $\Delta y_{\min}$ represent the minimum detectable translations.
The minimal detectable motion in the z-direction involves complex relationships with camera parameters and is particularly sensitive to distance, as detectability decreases with increasing depth. However, since the selected VO approach employs additional depth data from the ground truth execution (cf. Section 3.2.1), these are effectively addressed through additional depth information of the scene.

Field of View

The ability of camera systems to detect angular variations of its position is primarily limited by the FoV parameter, Θ_{max}. The FoV can be modified for cameras with interchangeable lenses by changing the lens's focal length, where the FoV is calculated using Equation (3.2).

$$\Theta_{\text{horiz}} = 2\arctan\left(\frac{\text{sensor width}}{2\cdot\text{focal length}}\right), \quad \Theta_{\text{vert}} = 2\arctan\left(\frac{\text{sensor height}}{2\cdot\text{focal length}}\right) \tag{3.2}$$

With the resolution, the FoV can now be divided into the least detectable angle rays, leading to a pixel difference.

$$\alpha_{min} = \frac{\Theta_{\text{horiz}}}{\text{resolution}_{\text{width}}} \tag{3.3}$$

This minimal angle indicates the smallest rotation around the x and y-axes at which the camera can still detect the error.
For an error introduced by tilting around the remaining z-axis, the position of possible environment features in the frame determines the possible minimum rotation angle. Assuming uniformly distributed global features, the features projected at the frame edge have a higher influence on the detectability of the camera rotation around z.

Frame Rate

The frame rate plays a critical role in the ILC's performance by determining the trajectory's sampling density. Higher frame rates enable the capture of more waypoints within a given trajectory duration, resulting in finer motion discretization and smoother execution.
The maximum achievable frame rate (FPS) can be calculated using Equation (3.4), which considers both the data transfer rate and chosen resolution, alongside the required exposure time. The data volume depends on the bit depth per pixel: grayscale images typically require 8 bits per pixel, while color images (RGB) need 24 bits per pixel.

$$\text{FPS}_{\text{max}} = \frac{1}{\text{Exposure Time} + \frac{\text{Resolution}\cdot\text{Bits per Pixel}}{\text{Data Transfer Rate}}} \tag{3.4}$$

The frame rate limitations directly translate into constraints on detectable motion. When generating depth data, sufficient feature matches between consecutive frames must be maintained, inherently restricting the camera's maximum velocity. This constraint particularly affects rotational movements around the x and y axes, with the maximum allowable rotation speed determined by both the camera's frame rate and FoV. For a specified percentage of frame overlap, the maximum rotational velocity around the x and y axes can be calculated using:

$$\omega_{max} = \text{FoV}\cdot\text{FPS}_{\text{max}}\cdot(1-\text{overlap})\,. \tag{3.5}$$

Similar to the minimum detectable translation, the maximum allowable translational camera motion is constrained by the depth of detected features in the scene. Due to

perspective projection, features closer to the camera experience larger apparent motion than those further away.

Motion Blur

Motion blur is an additional limitation, introduced when the camera's exposure time is too long relative to the camera motion. The long exposure time will cause unintended illumination of additional pixels, leading to a distortion of the image's features. This distortion changes the descriptor of the feature and deteriorates the matching between different image frames.
The maximum achievable angular motion, ensuring motion blur does not occur, is shown in Figure 3.3. The critical camera properties influencing the limitation of angular motion blur include resolution, focal length, and exposure time. These findings are specific to the PhotonFocus camera and are based on data presented in Table 3.2, assuming a constant exposure time of 5 ms.

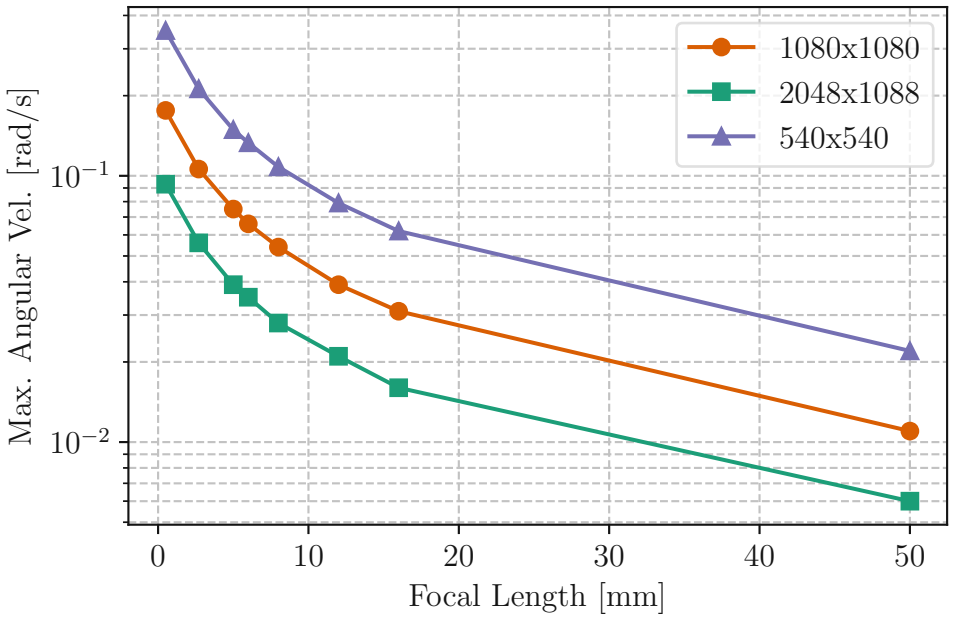


Figure 3.3: Depiction of how motion blur impacts the achievable maximum angular velocity employing the selected PhotonFocus camera, taking into account an exposure time of 5 ms, a range of focal lengths, and different resolutions.

Selected Camera

Considering the outlined criteria, the Photonfocus DR1-D2048x1088-192-G2 camera is selected, which is shown in Figure 3.4. This camera features a high resolution and an extensive active optical area, establishing a solid foundation for achieving a wide FoV. The camera specifications and calculated VO constraints are summarized in Table 3.2. The GigE interface provides easy integration with ROS through the CAMERA_ARAVIS library, simplifying camera management by allowing convenient adjustments to parameters such as exposure time, digital and analog gain, and black level. Additionally, the

Table 3.2: Photonfocus DR1-D2048x1088C-192-G2 Specifications

Property	**Value**
Resolution	2048 × 1088 px
Active optical Area	11.26 mm × 5.98 mm
Pixel Size	5.5 µm × 5.5 µm
Interface	GigE
Shutter Mode	Global Shutter
ext. Focal Length	6 mm
Δx_{min}	0.003 mm
Δy_{min}	0.003 mm
α_{min}	0.0364°
ω_{max}	239.25 ° s^{-1}

library supports software-based triggering of the camera using a dedicated ROS node. This node interacts with the robot's status, ensuring that camera signals are always triggered at the same moments for each waypoint in each iteration.

Figure 3.4: Photon Focus DR1-D2048x1088C-192-G2 color camera [87].

3.2 Motion Error Estimation

The motion error estimation is a central part of the thesis. Its task is to identify deviations in the trajectory of the robot's end-effector introduced by kinematic and non-kinematic errors. These detected deviations are utilized as sensor input for the update phase of the ILC process.
To estimate these errors, a camera is mounted on top of the end-effector in an eye-in-hand configuration. The generic environment of the robot is then used to detect distinct features that establish the baseline of the motion deviations in a trajectory.

For a given trajectory on each waypoint i, a pose error estimation is performed by comparing the current waypoint images against the ground truth trajectory reference

images via monocular VO (cf. Section 2.7).
The influence of an error on the individual waypoints of a trajectory can be seen in Figure 3.5. Here, the trajectory deviations between the ground truth run and iteration j result in spatial displacement of the camera coordinate system $\mathcal{C}$ at each waypoint.

While monocular VO enables precise pose deviation detection, the inherent scale

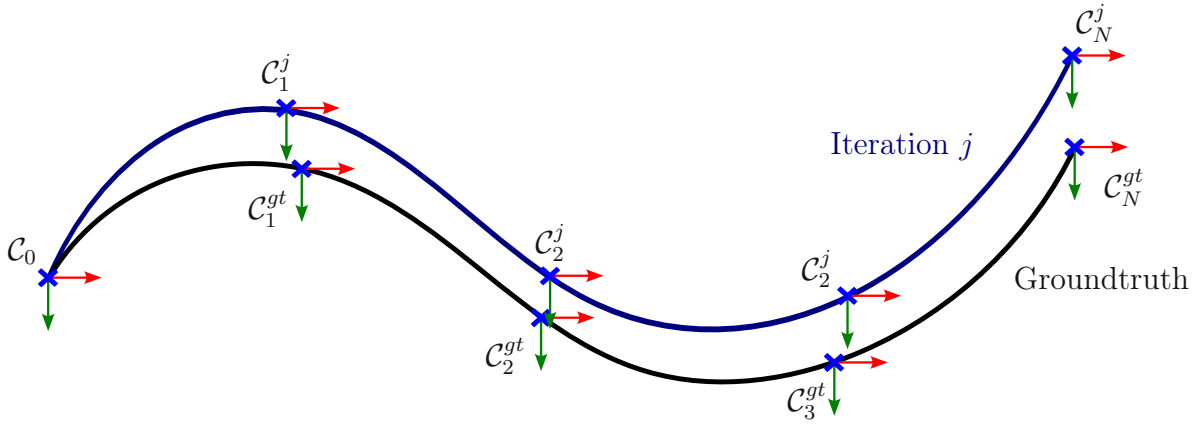


Figure 3.5: 2D representation of the deviation between the ground truth trajectory and the trajectory at iteration j. Where $\mathcal{C}$ are the camera origins at each waypoint $i \in [0, N]$ with a similar initial starting position $\mathcal{C}_0$

ambiguity challenges accurate translation estimation [45]. To address this limitation, the known robot poses and images from the ground truth run are utilized to generate depth data, providing the necessary scale information.

3.2.1 Depth Estimation

The global feature map is built using the recorded images and their corresponding robot poses obtained from the ground truth trajectory execution. The map incorporates environment features extracted and matched across all images taken during the ground truth trajectory execution (cf. Figure 3.6).
These environment features are extracted using Scale-Invariant Feature Transform (SIFT) detector, transforming them into image features, each comprising a keypoint (its position in the image frame) and a descriptor (cf. Section 2.7.1).
The map is initially populated with all image features from the first waypoint recording, where each image feature is directly mapped to an environmental feature in the global map. The image features from subsequent waypoint recordings are matched against the global map using a k-Nearest Neighbours (kNN) matcher.
The matching process adapts its strategy based on whether the best match corresponds to an environment feature with single or multiple descriptor entries. If the best match is to a feature with only one descriptor, this indicates that it is a newly added feature. In this case, the traditional Lowe's ratio test (cf. Section 2.7.1) is applied by comparing the distances of the two best matches to ensure that they are sufficiently different. For features with multiple existing descriptor entries, it calculates the average distance to all descriptors of the currently best-matched feature and compares this against the best

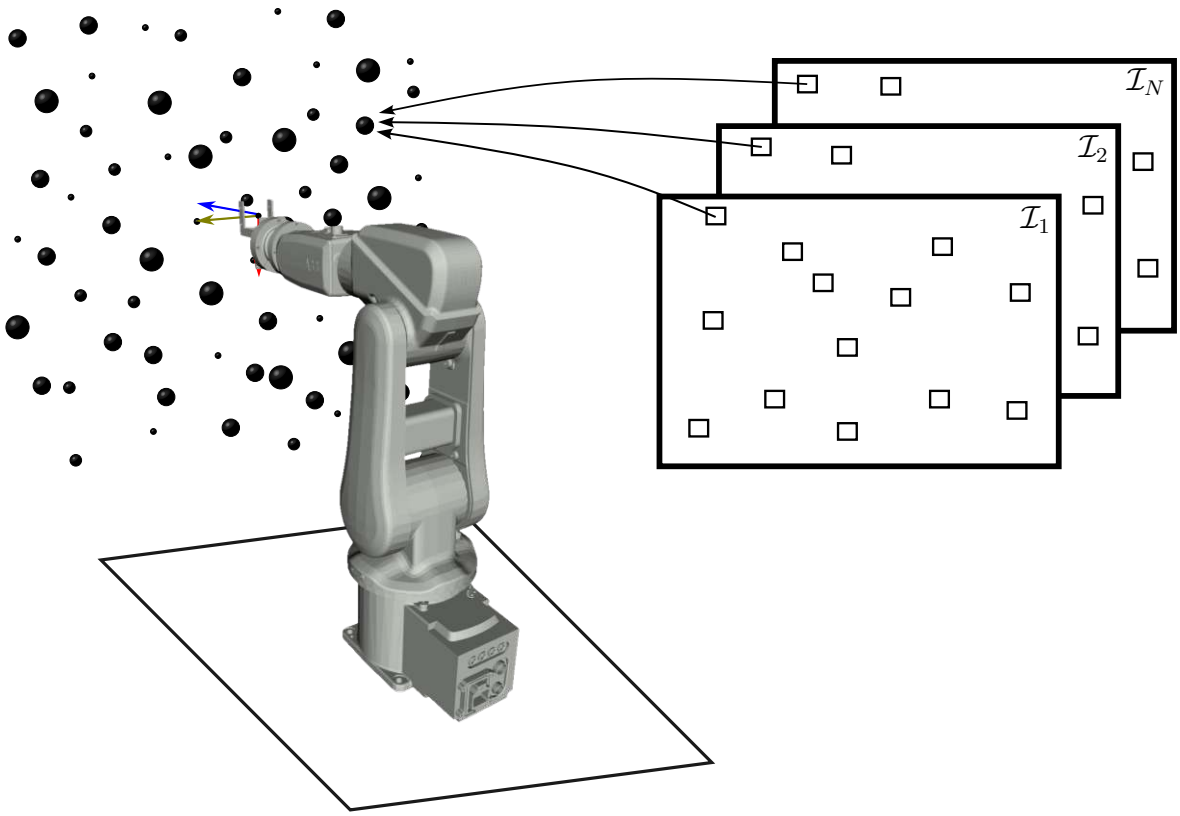


Figure 3.6: Visualization of the relationship between the environment features in the global feature map (spheres), and the image features in the individual frames of the waypoints (squares). The map features are created via triangulation, using multiple views of the same environment utilizing the recorded image $\mathcal{I}_i$ from the waypoints $i = 1, 2...N$.

distance to any other environment feature's descriptors.
This approach increases the likelihood that new descriptors will be matched with similar, pre-existing descriptors, helping to reduce false associations and mismatches.
The process is parallelized using OpenMP, allowing efficient handling of long trajectories with numerous waypoints while maintaining data consistency through proper synchronization.

Successfully matched image features are linked to the corresponding environmental features in the map, along with the robot's pose at the relevant waypoint. If an image feature fails to find a match, it is added as a new environmental feature to the global map.
This approach enables the map to expand over time, incorporating new features that can be used for future matching and ongoing refinement.

Given a global feature map with multiple environment features, these features will contain multiple waypoint image matches and their feature information. For each environment feature in the map, the image information is used to perform a triangulation process to compute a depth estimate in the global environment (see Section 2.6.1).

This way, multiple depth estimates are generated for each feature through various waypoint-view combinations.
To minimize noise from the images and incorrect matches, the depth estimation is then filtered using a geometry-based Random Sample Consensus (RANSAC) algorithm. This method constructs a fundamental matrix from the keypoint entries of the environment features, eliminating those that fall outside the acceptable tolerance range. The final depth value for each environment feature is then calculated as the average of the filtered depth estimates.

3.2.2 Visual Odometry

A monocular VO algorithm is employed to estimate pose deviations at each waypoint inspired by [88]. This specific algorithm is chosen for its ability to handle sparse depth information, incorporating additional depth data from the reference trajectory run generated by the depth estimation (cf. Section 3.2.1). The features used in this process are detected and matched using SIFT, ensuring robust feature correspondence between consecutive frames. This approach is particularly suitable since only sparse depth data from the ground truth recordings are available.

Mathematical Model

The algorithm uses the camera motion derived as the rigid body transformation in Equation (3.6), where $\mathbf{R}$ is the rotation matrix and $\mathbf{t}$ is the translation vector. A tracked environment feature matched from a waypoint image pair of the iteration $\mathcal{I}^j$ and the corresponding ground truth $\mathcal{I}^{gt}$, denoted as $\mathbf{p}_i^{gt}$ and $\mathbf{p}_i^j$, is expressed in coordinates $[x_i^j, y_i^j, z_i^j]^T$, with the origin positioned at the optical center of the camera $\mathcal{C}^{gt}$ and $\mathcal{C}^j$. For features with unknown depth, the coordinates are normalized to $\bar{\mathbf{p}}_i^j = [\bar{x}_i^j, \bar{y}_i^j, 1]^T$.

$$\mathbf{p}_i^j = \mathbf{R} \cdot \mathbf{p}_i^{gt} + \mathbf{t} \tag{3.6}$$

As the depth of the coordinate at iteration j is not known, Equation (3.6) can be rewritten as

$$z_i^j \bar{\mathbf{p}}_i^j = \mathbf{R} \cdot \mathbf{p}_i^{gt} + \mathbf{t}. \tag{3.7}$$

By combining the rows of the matrix, this can be simplified into the following two equations

$$\left(\mathbf{r}_1^T - \bar{x}_i^j \mathbf{r}_3\right) \mathbf{p}_i^{gt} + t_1 - \bar{x}_i^j t_3 = 0, \tag{3.8}$$

$$\left(\mathbf{r}_2^T - \bar{y}_i^j \mathbf{r}_3\right) \mathbf{p}_i^{gt} + t_2 - \bar{y}_i^j t_3 = 0 \tag{3.9}$$

where $\mathbf{r}_h^T$ and t_h, for $h \in \{1, 2, 3\}$, represent the h-th row of the rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$, respectively.
For environment features without existing depth estimation from the ground truth mapping, Equation (3.6) can be rewritten as

$$z_i^j \bar{\mathbf{p}}_i^j = z_i^{gt} \mathbf{R} \cdot \bar{\mathbf{p}}_i^{gt} + \mathbf{t}. \tag{3.10}$$

Both coordinates are subsequently normalized and combined into 3.11, eliminating z_i^j and z_i^{gt}.

$$[-\bar{y}_i^j t_3 + t_2, \bar{x}_i^j t_3 - t_1, -\bar{x}_i^j t_2 + \bar{y}_i^j t_1]\mathbf{R}\bar{\mathbf{p}}_i^{gt} = 0 \tag{3.11}$$

The motion is now solved by stacking the residual equations from (3.8) and (3.9), or by decomposing the matrix equation (3.11) into its three scalar equations, into the nonlinear function,

$$\mathbf{f}([\mathbf{t};\boldsymbol{\theta}]) = \boldsymbol{\epsilon} \tag{3.12}$$

creating a vector $\boldsymbol{\epsilon} \in \mathbb{R}^{2m+3n\times 1}$ filled with the residuals. Here, m and n represent the number of features with depth and without depth values.

Optimization

In order to find values for $[\mathbf{t},\boldsymbol{\theta}]$ which minimize the error of the projection between the two poses, the following optimization problem is formulated:

$$\underset{\mathbf{t},\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\mathbf{t},\boldsymbol{\theta}) = \underset{\mathbf{t},\boldsymbol{\theta}}{\operatorname{argmin}}\, \frac{1}{2}\|\mathbf{f}([\mathbf{t};\boldsymbol{\theta}])\|_2^2. \tag{3.13}$$

To solve this optimization problem, the Levenberg–Marquardt algorithm (LM) is utilized. The LM algorithm is an iterative optimization method that combines the advantages of the Gauss-Newton method and gradient descent.
At each iteration, the algorithm computes an update for the parameters $[\mathbf{t},\boldsymbol{\theta}]$ by solving the linearized system

$$\Delta[\mathbf{t};\boldsymbol{\theta}] = -(\mathbf{J}^\top\mathbf{J} + \lambda\mathbf{I})^{-1}\mathbf{J}^\top\boldsymbol{\epsilon}. \tag{3.14}$$

Here, $\mathbf{J} = \frac{\partial \mathbf{f}}{\partial[\mathbf{t};\boldsymbol{\theta}]}$ is the Jacobian matrix of the residuals $\boldsymbol{\epsilon}$, and λ is the damping parameter. Once the parameter update $\Delta[\mathbf{t};\boldsymbol{\theta}]$ is computed, the parameters are updated iteratively:

$$[\mathbf{t};\boldsymbol{\theta}]_{k+1} = [\mathbf{t};\boldsymbol{\theta}]_k + \Delta[\mathbf{t};\boldsymbol{\theta}]. \tag{3.15}$$

The process repeats until one of the convergence criteria is met. These criteria are the magnitude of the parameter update $\|\Delta[\mathbf{t};\boldsymbol{\theta}]\|$, the falling below a threshold, the relative change in the cost function being small, or the maximum number of iterations being reached.
In cases where convergence fails for a particular waypoint, the system adopts a fallback strategy by using the motion estimation from the previous waypoint as an approximation, allowing the next iteration to proceed with motion estimation based on this substitute value.

Implementation

This optimization process is performed for each image waypoint pair using the CERES-SOLVER. Since the ILC update step is performed after the trajectory has been executed, individual waypoints can be calculated in parallel using multithreading with OPENMP, significantly reducing processing time. CERESSOLVER streamlines the optimization process by automatically computing derivatives and related functions once the residual

template functions are defined. Given that the optimization is performed offline, computational efficiency becomes secondary to the convergence accuracy. This allows for the implementation of stringent convergence criteria as shown in Table 3.3, without real-time performance constraints.

Table 3.3: CeresSolver stopping criteria

Parameter	Symbol	Value
Function tolerance	ϵ_1	1e−7
Gradient tolerance	ϵ_2	1e−11
Parameter tolerance	ϵ_3	1e−8

3.2.3 Error Calculation

The translational and rotational deviation of the camera $[\hat{\mathbf{t}}; \hat{\boldsymbol{\theta}}]$, now describes the end-effector error between the ground truth pose and the estimated pose. This error is now converted to the individual joint errors to be usable in the ILC.
To achieve this, the known ground truth pose of the camera $\mathbf{P}_{\mathcal{W}}^{\mathcal{C}_{gt}}$ is transformed using the estimated error resulting in the detected transformed pose $\hat{\mathbf{P}}_{\mathcal{W}}^{\mathcal{C}_j}$.
Therefore, the vector representation $\hat{\boldsymbol{\theta}}$ is transformed into the rotation matrix $\hat{\mathbf{R}}$ and inserted together with $\hat{\mathbf{t}}$ into the homogeneous transformation matrix $\hat{\mathbf{T}}_{\mathcal{C}^{gt}}^{\mathcal{C}_j}$.
The now transformed pose $\hat{\mathbf{P}}_{\mathcal{W}}^{\mathcal{C}_j}$ is then inverted to the joint angle using the Jacobian Method implemented in MoveIt! and the known robot kinematic model (see Section 2.2.3). This process is described in the equations 3.16, 3.17, and 3.18.

$$\hat{\mathbf{T}}_{\mathcal{C}^{gt}}^{\mathcal{C}_j} = \begin{bmatrix} \hat{\mathbf{R}} & \hat{\mathbf{t}} \\ 0 & 1 \end{bmatrix} \tag{3.16}$$

$$\hat{\mathbf{P}}_{\mathcal{W}}^{\mathcal{C}_j} = \hat{\mathbf{T}}_{\mathcal{C}^{gt}}^{\mathcal{C}_j} \cdot \mathbf{P}_{\mathcal{W}}^{\mathcal{C}_{gt}} \tag{3.17}$$

$$\hat{\mathbf{q}}_j = f_{\text{kin}}^{-1}(\hat{\mathbf{P}}_{\mathcal{W}}^{\mathcal{C}_j}) \tag{3.18}$$

The difference between the joint angles of the ground truth and the calculated inverse of the transformed pose is now the error used to update the ILC, defined as

$$\mathbf{e}_j^q = \hat{\mathbf{q}}_j - \mathbf{q}_{gt}. \tag{3.19}$$

This error now describes the joint error of all axis for one single waypoint of the ILC iteration j.
To improve the accuracy of the inverse kinematics calculation, the search range $[q_j^{\min}, q_j^{\max}]$ is iteratively reduced over time:

$$[q_j^{\min}, q_j^{\max}] = \pm \frac{\pi}{1 + 0.2 \cdot n_{\text{iterations}}} \tag{3.20}$$

where $n_{\text{iterations}}$ represents the current iteration number.

3.3 Iterative Learning Control

For the control of the industrial robot, a PD-type ILC approach is employed for its ability to operate without an explicit dynamic model of the robot, as well as being particularly suitable for non-linear systems [34]. This choice avoids the challenges of developing accurate dynamic models, which often require detailed manufacturing information that may not be available. The implemented ILC state machine architecture, consisting of the state machine sequence, ROS action servers, and data handling through ROSBAGS, is detailed in Figure 3.7.

The complete ILC process structure is integrated as a state machine into ROS using

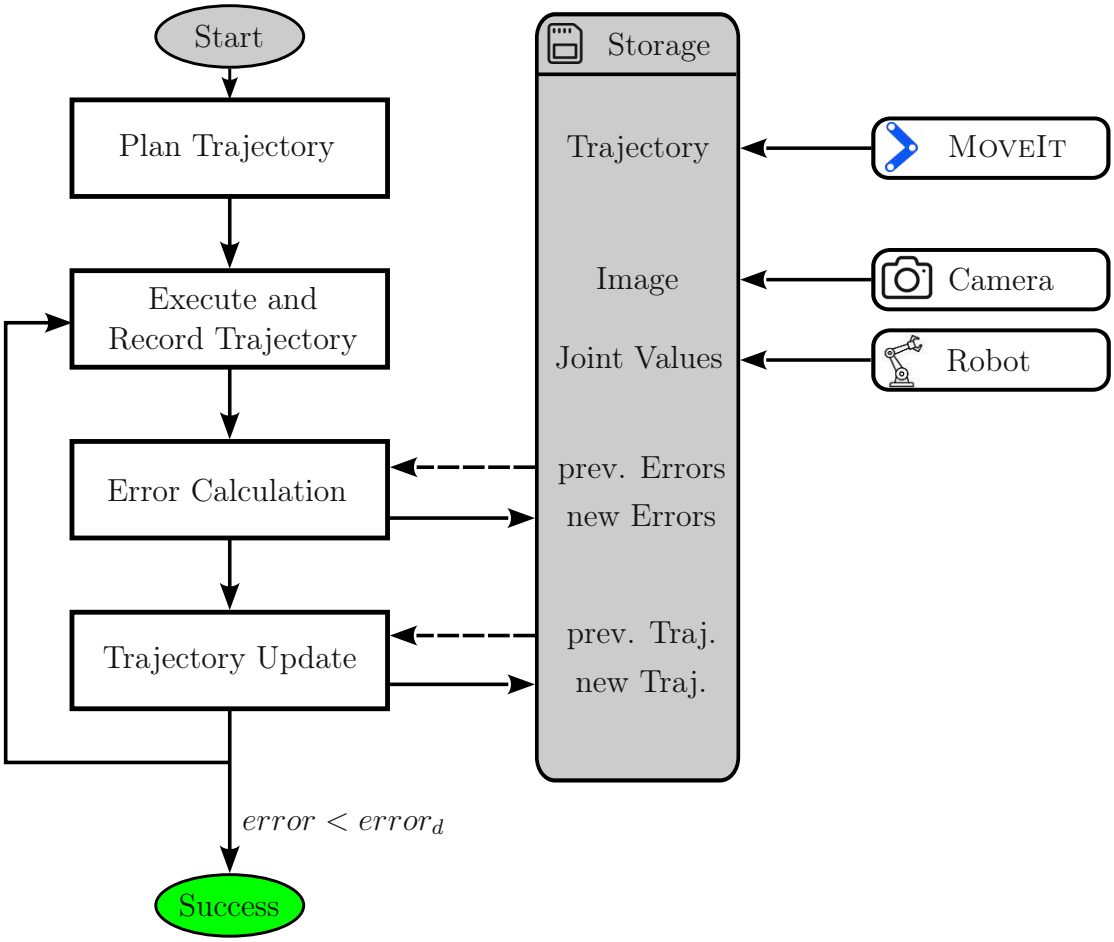


Figure 3.7: Architecture of the ILC state machine implementation. The left side displays the state sequence, where each state executes specific tasks through dedicated ROS action servers. States transition sequentially upon successful task completion. Data from external additional components (camera, robot, and MoveIt-Planner) shown on the right side are logged and retrieved via ROSBAGS throughout the process. This sequence iterates continuously, refining the trajectory until the measured error falls below a desired minimum error value $error_d$ or the maximum number of iterations is reached.

the `ros_smach` package. The state machine calls modularly implemented ROS action servers for trajectory planning, handling the simultaneous robot execution and image

recording, and applying PD-type trajectory updates. Data from previous iteration executions, such as the waypoint images and the corresponding joint values, are saved into ROS-BAG files as memory management.
The employed trajectory update $\Delta \mathbf{u}_j$ can be expressed in the lifted system as

$$\Delta \mathbf{u}_j = \mathbf{K}_p \cdot \mathbf{e}_j^q + \mathbf{K}_d \cdot \frac{\Delta \mathbf{e}_j^q}{\Delta t} \tag{3.21}$$

with the gain matrices $\mathbf{K}_p$ and $\mathbf{K}_d$.
This update is then filtered using a low-pass filter by applying a filter value α to the current and previous update obtaining the new trajectory

$$\mathbf{u}_{j+1} = \mathbf{u}_j + [\alpha \cdot \Delta \mathbf{u}_j + (1 - \alpha) \cdot \Delta \mathbf{u}_{j-1}]. \tag{3.22}$$

The joint errors for the update step are calculated as in Section 3.2.3 mapping the end-effector derivation to the angular joint values usable for the ILC.
The error calculation process, which uses the inverse kinematic model to convert end-effector deviations into joint values, technically resembles a model-based ILC approach. However, since the compensation needs to address both kinematic and dynamic deviations, and the kinematic model alone cannot account for dynamic effects, the ILC is implemented and tuned as a PD controller instead. The controller gains are experimentally determined through systematic testing, with particular attention to minimizing overshoot while maintaining convergence speed.

CHAPTER 4

Simulational Validation and Real-World Testing of the vision-based Iterative Learning Control System

In order to evaluate the overall performance of the Iterative Learning Control (ILC) system mitigating the end-effector deviations, experiments are performed with a given trajectory in the workspace. Therefore, the broader investigation for the error estimation is done using simulation, followed by a real-world application using a 6-axis robot.
The evaluation is structured into three main sections: First, the methodology describes the experimental setup and procedures for both simulation and real-world tests. This is followed by a comprehensive presentation of the results from both test setups. Finally, a discussion section analyzes the findings and their implications and limitations for practical applications.

4.1 Methodology

In this section, the system and its components introduced in Chapter 3 are further specified for the upcoming simulations and experiments.
For this purpose, the hardware utilized for the simulation and experiment is discussed first. Subsequently, the robot, its control, the calibration of the camera, and the connection of these two components to the ROS environment are described.
Finally, the implementation of the error estimation simulation is presented. Furthermore, also the performance metrics used for the analysis and the experimental execution of the ILC is described.

4.1.1 Computer Specifications

The simulation, as well as all software implementations, are performed on a single machine. To provide a complete overview of the simulation and experiment setup for future comparisons, the hardware specifications are listed in Table 4.1.

Table 4.1: PC Component Specification

Part	Name
CPU	Intel Core i9-14900k
GPU	Gainward GeForce RTX 4070 Ghost 12GB GDDR6X
RAM	Kingston FURY DIMM 32GB DDR5-6000
Mainboard	ASUS Prime Z790-P

4.1.2 Robot

The industrial robot ABB-IRB-120 is chosen as the robotic manipulator for the experimental investigation of the ILC. This robot has six rotational axes arranged similar to an anthropomorphic human arm and allows movements of the end-effector in all six degrees of freedom.
The robot deploys an angular encoder capable of an angular accuracy in each joint up to 174 μrad, sufficiently enough for the upcoming experiments. Its specifications, such as angular and Tool Center Point (TCP) limitations, are given in Table 4.2.
To communicate with the employed Robot Operating System (ROS) framework, send trajectory updates, and receive current joint positions, the open source software OPEN_ABB[1] is used. The OPEN_ABB software, installed directly on the ABB-IRB-120 robot's IRC5 controller as well as a ROS package, enables this communication by providing various ROS services and publishers.

Table 4.2: ABB IRB120 Specifications [89]

Angular Specifications		
Movement	Joint values	Max. angular velocity
Axis 1	+165° to -165°	250 ° s^{-1}
Axis 2	+110° to -110°	250 ° s^{-1}
Axis 3	+70° to -110°	250 ° s^{-1}
Axis 4	+160° to -160°	320 ° s^{-1}
Axis 5	+120° to -120°	320 ° s^{-1}
Axis 6	+400° to -400°	420 ° s^{-1}
Spatial Specifications		
Accuracy	0.01 mm	
Max. TCP velocity	6.2 m/s	
Max. TCP acceleration	28 m/s^2	

[1] https://github.com/robotics/open_abb

4.1.3 Camera Calibration

For the experimental setup, the PhotonFocus R1-D2048x1088C-192-G2 color camera (Figure 3.4) is attached to the end-effector of the deployed robot.
The camera is calibrated in order to minimize the distortion of the camera for the motion estimation. For this purpose, the `calibrate_camera` package from ROS is utilized. This package can easily determine the internal camera parameters with only the launch parameters for the camera's video stream topic and a provided chessboard pattern. After launching, subsequent images of the chessboard are taken in different positions and orientations within the Field of View (FoV) of the camera. These recorded images are processed on the fly for calibration using OpenCV (cf. Section 2.5.3), providing the intrinsic camera at the end.

4.1.4 Simulation: Motion Estimation

A simulation is carried out using a point cloud to determine the minimum possible recognizable motion estimation and to determine various influencing factors.
This point cloud is generated from randomly placed 3D points in a predefined area, which are then projected onto an artificial camera frame using the `projectPoints()` function of OpenCV.
The deviating movement of the end-effector is simulated by the extrinsic camera parameters $\mathbf{R}_{cam}$ and $\mathbf{t}_{cam}$, which are passed to the function and represent the displacement of the camera to the global frame of the point cloud.
In order to determine the displacement using the Visual Odometry (VO) of Section 3.2.2, the points are projected once without displacement and once with displacement parameters, creating simulated feature points of a scene from two different views. In addition, Gaussian noise is added, which simulates realistic measurement uncertainties by applying random perturbations to the projected pixel coordinates of each 3D point using a zero-mean normal distribution with configurable standard deviation. This influence of the noise on the individual keypoints is illustrated in Figure 4.1.
The features are then compared with the limits of the image size corresponding to the camera used in the real experiment. If necessary, the features are discarded.
Since the VO works with a mixture of matched features with and without depth data, a percentage of the projected reference features are supplemented with the respective depths, simulating the depth estimation of Section 3.2.1 for the necessary ground truth trajectory.
In total, three variables can be modified freely to evaluate the quality of the motion estimation. These variables are the number of 3D points created in the defined scene, the standard deviation of the noise, and the number of features used with additional 3D depth data.

Due to the possible movement deviation of the camera in all six degrees of freedom, it is time-consuming to examine all combinations of different displacements. For this reason, the three translational and rotational movements are each grouped together and always moved by the same amount in their three directions.
Thus, three different types of movement simulations are performed. The isolated

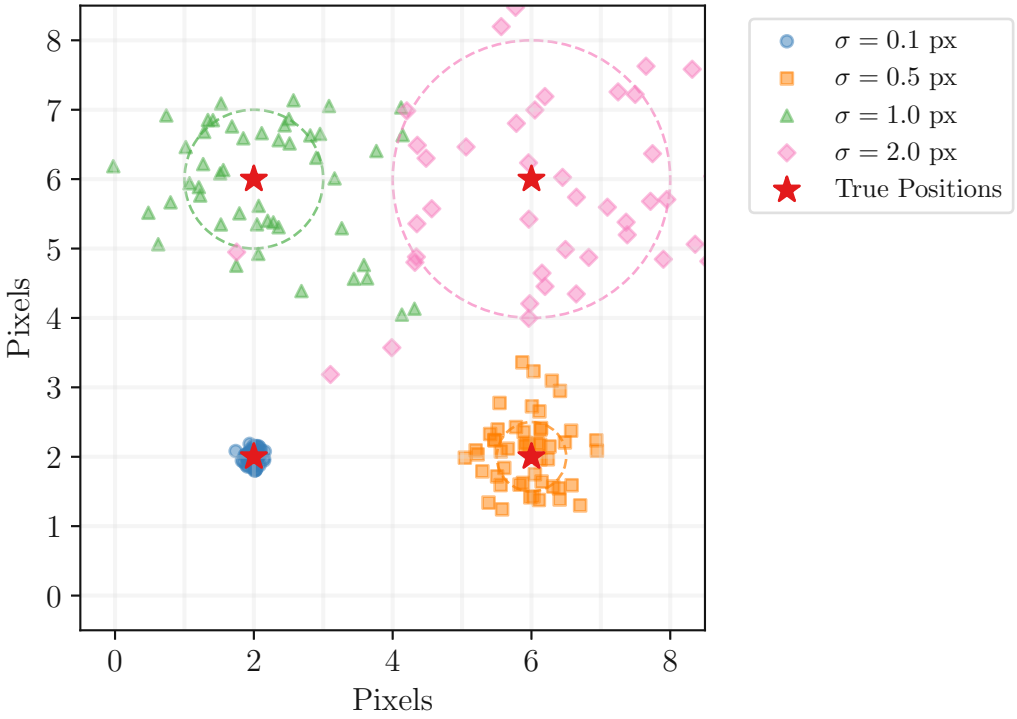


Figure 4.1: Feature point projection uncertainty under varying levels of Gaussian noise. The true keypoint positions (red stars) are shown with their corresponding noisy measurements, simulated with standard deviations σ ranging from 0.1 to 2.0 pixels. Dashed circles represent the one standard deviation boundary, illustrating the increasing spatial dispersion of measurements with higher noise levels.

movements in rotation and translation, as well as their combination, i.e., simultaneous rotation and translation.

To evaluate the motion estimation's suitability for ILC, the analysis focuses on the Signal-to-Noise Ratio (SNR). This metric is chosen as it directly relates to the ILC's convergence requirements, where the errors of the estimation itself must be sufficiently small relative to the true motion deviations that need to be detected. For each parameter configuration (cf. Table 4.3), 100 trials are conducted to ensure statistical reliability. A theoretical minimum valid SNR threshold is established as a baseline for acceptable performance in ILC applications.

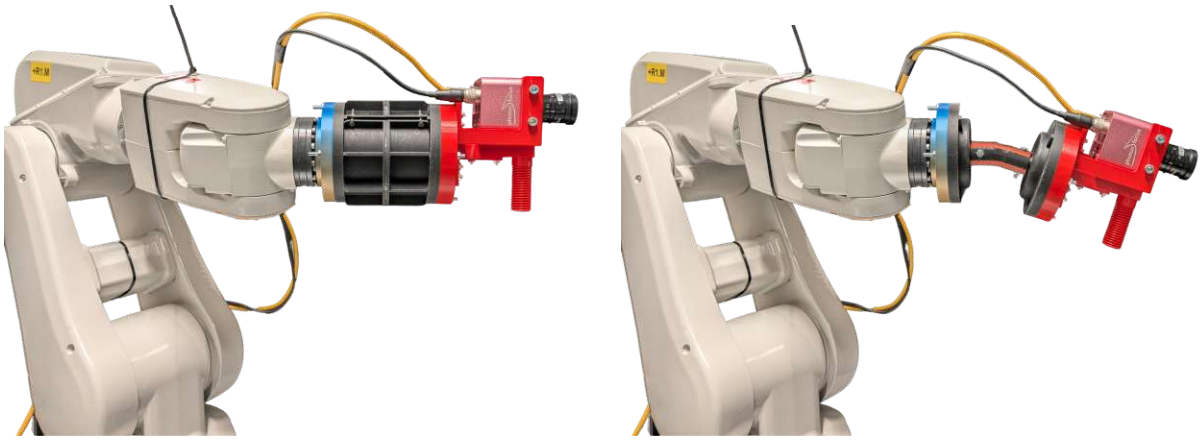
4.1.5 Experiment: Iterative Learning Control

The experimental validation of the proposed vision-based ILC utilizes an eye-in-hand configuration, consisting of an ABB IRB 120 industrial robot with 6 degrees of freedom (presented in Section 4.1.2) and a PhotonFocus camera (described in Section 3.1.2). Due to the robot's inherent rigidity and minimal end-effector flexibility, a custom 3D-printed camera attachment is used to investigate the end-effector behavior under high flexibility conditions. The attachment introduces controlled end-effector flexibility, which can

Table 4.3: Simulation parameters and their ranges used for evaluating the motion estimation algorithm.

Variable	Parameters
X range	[-5, 5] m
Y range	[-5, 5] m
Z range	[0, 10] m
Depth ratios	{0, 20, 40, 60, 80, 90, 100}%
Translation magnitudes	{0.0005, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2} m
Rotation magnitudes	{0.0005, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2} rad
Number of 3D points	{100, 200, 500, 1000, 2000, 5000, 10000}
Noise levels σ	{0, 0.1, 1, 2, 5, 10} pixels

be systematically reduced through additional housing components to achieve a rigid robot-to-camera transformation. The rigid configuration is essential for the ground truth reference runs, enabling the acquisition of ground truth data for the depth estimation and VO. The complete setup with the flexibly mounted camera is shown in Figure 4.2. To evaluate the setup's capability to compensate for both kinematic and dynamic



(a) (b)

Figure 4.2: Eye-in-hand camera attachment utilized for the ILC experiments. Figure a) shows the rigid connection serving as a reference in the ILC. In Figure b), the connection is flexible, resulting in kinematic and non-kinematic deviations at the end-effector.

impacts on the end-effector accuracy, a trajectory with two transition points of high gravitational and dynamic influence is selected. The path forms a triangle on the x-axis plane, traversing between three points (A, B, and C) as shown in Figure 4.3.
The chosen trajectory, which includes 23 waypoints spread throughout the robot's workspace, forms the foundation for all the different experimental setups. The quantity of waypoints is determined by considering the camera's limiting trigger capability and the required smoothness and sampling of the given maximum trajectory velocity.

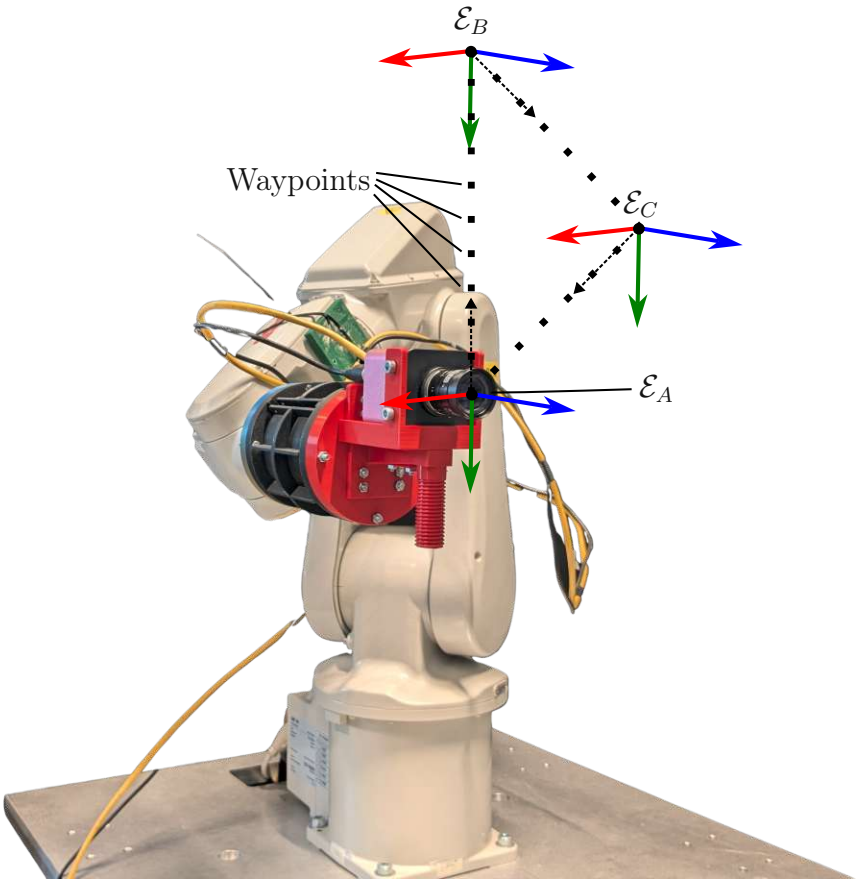


Figure 4.3: ABB IRB 120 robot arm with mounted camera, showing the employed end-effector trajectory and waypoints between the traversing poses $\mathcal{E}_A$, $\mathcal{E}_B$, and $\mathcal{E}_C$.

To investigate the algorithm's performance across different operating conditions, the experiments are conducted at multiple end-effector velocities. These velocities present the robot's maximum joint velocity override setting and are denoted as `trial_10%` to `trial_40%`.
The actual end-effector velocity is calculated using MoveIt's forward kinematics and the recorded joint data are listed in Table 4.5. The upper limit is constrained by the camera's image acquisition timing and triggering capabilities.

Baseline Performance Assessment

The first experimental setup evaluates the motion estimation algorithm's baseline accuracy under ideal conditions, with camera housings remaining mounted for both ground truth and estimation runs to eliminate induced deviations. The investigation of motion detection in the absence of motion is important because the ILC aims to converge precisely this non-motion. The accuracy of the ILC can, therefore, be traced back to the accuracy with which no movement can be detected.
To obtain a valid statistical statement, testing consists of all four velocities trials for 20 iterations each.

Camera Mount Characteristics

The second experiment analyzes mount-specific influences by comparing the rigid ground truth and the flexible mount configurations. System performance is evaluated across 200 iterations at the maximum trajectory speed without ILC trajectory updates to isolate mount-related effects.

Error Deviation Compensation Assessment

The third experiment evaluates ILC performance across the four trial velocities (`trial_10%`, `trial_20%`, `trial_30%`, and `trial_40%`) using motion estimation errors for trajectory updates. For this, conservative control parameters from Table 4.4 are chosen, where the identity matrix $\mathbf{I}$ is employed as a scaling factor to facilitate parameter setting. The system is run for 100 iterations to ensure sufficient data for convergence analysis while maintaining trajectory smoothness.

Table 4.4: Experiment: ILC parameter

Parameter	Values
$\mathbf{K}_p$	$\mathbf{I} \cdot 0.2$
$\mathbf{K}_d$	$\mathbf{I} \cdot 0.002$
α	0.75

Dynamic Deviation Compensation Assessment

Since the error estimation cannot be compared by external means, it must nevertheless be demonstrated that dynamic errors can also be detected and, therefore, mitigated. These dynamic errors occur predominantly in the current setup due to inertia. For this reason, the waypoints of the trajectory that are exposed to high accelerations are analyzed separately, as the dynamic effects are most significant in these cases.
The maximum acceleration is computed through numerical differentiation of the end-effector's position, derived from forward kinematics applied to 100 iterations of joint data recordings. The resulting end-effector trajectory accelerations are visualized in Figure 4.4. Based on this acceleration profile, waypoints 7, 8, 14, and 15 are identified as peaks exhibiting the highest acceleration values, warranting further investigation.
To isolate the dynamic effects at these points, a mean kinematic position error due to static flexibility is calculated from the low-acceleration waypoints and subsequently subtracted from the selected high-acceleration waypoints, effectively isolating the position errors attributable to dynamic effects.

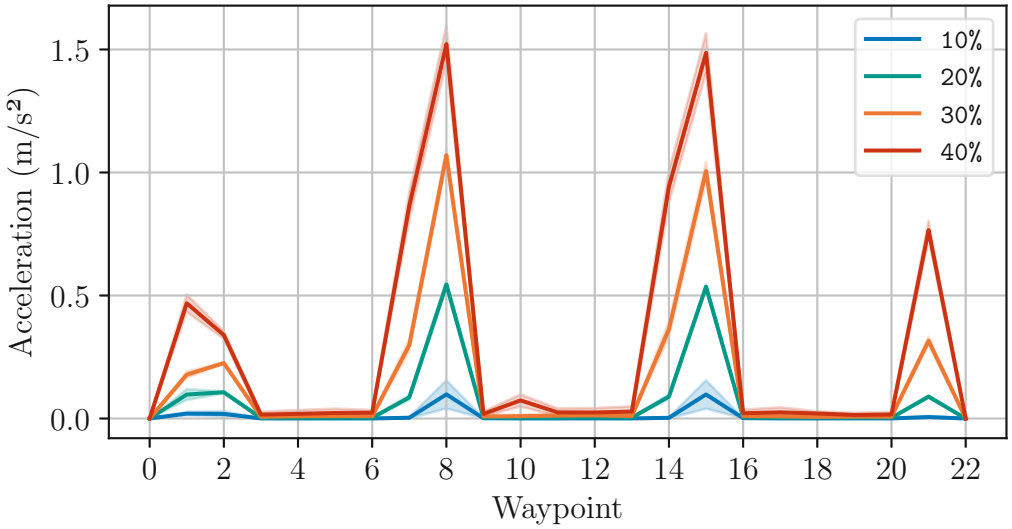


Figure 4.4: Mean end-effector acceleration profiles of the used trajectory. The values are calculated across 100 iterations of all four velocity trials (`10%-40%`), utilizing the kinematic model and recorded joint trajectory data.

4.2 Results

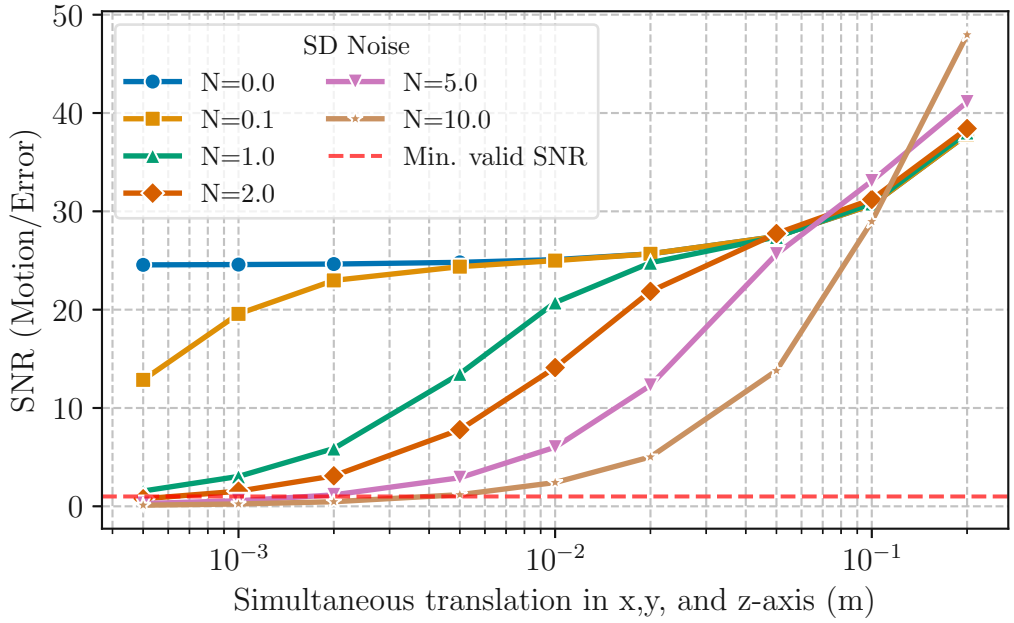
This section presents and analyzes both the simulation and experimental results of the vision-based ILC implementation. The results were obtained using the methods and parameters detailed in Section 4.1, and are presented in standardized plots to enable direct comparison.

4.2.1 Simulation: Motion Estimation

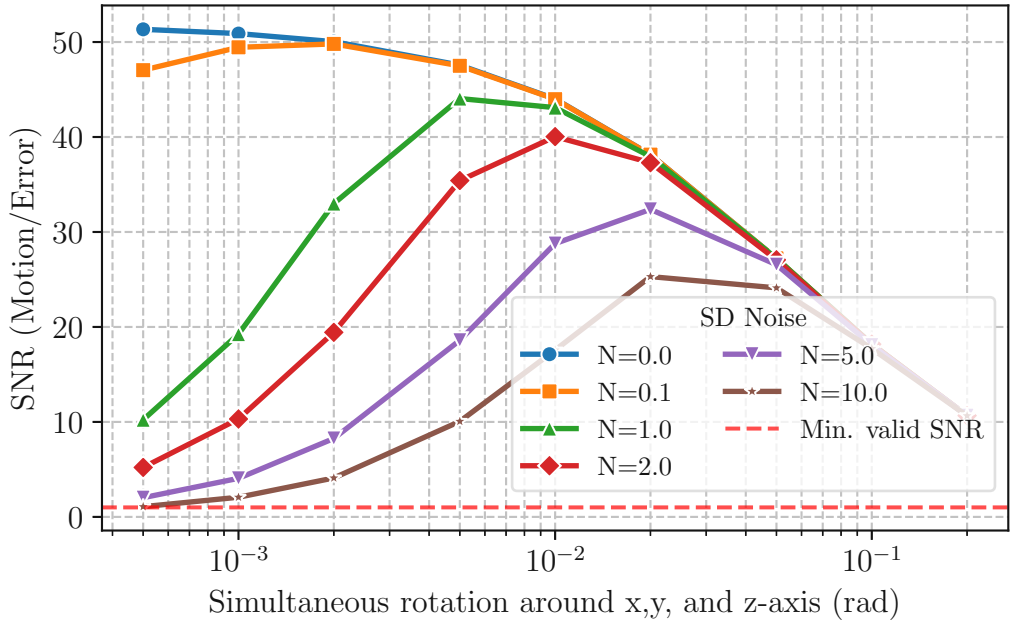
The SNR analysis reveals the relationship between estimation accuracy and motion magnitude under various conditions. The results are presented for three key influences: noise (Figure 4.5), depth (Figure 4.6), and feature count (Figure 4.7).

Noise Influence

The analysis in Figure 4.5, performed with 100% depth values, demonstrates that translational motion estimation is more sensitive to noise than rotational estimation. Noise with standard deviations exceeding two pixels results in SNR values below one, particularly at small motions below 1 mm.



(a)



(b)

Figure 4.5: Simulation results visualizing the camera motion estimation accuracy analysis under varying image noise, implemented as a zero-mean Gaussian distribution. The noise applies to the projected 2D coordinates of 3D scene points. The analysis presents the estimation performance for (a) simultaneous translation and (b) simultaneous rotation across all three axes (x, y, z), where N represents the standard deviation of the noise.

Depth Influence

The SNR data shown in Figure 4.6 shows the influence of the depth data on the ability to detect the specified deviations of the camera with an applied standard deviation noise of 1. The translational SNR becomes exponentially worse the less depth data is available. This is particularly visible in the translation, where the SNR for 100 % increased rapidly for higher motion magnitudes, whereas only a relatively stable SNR value of 2 is achieved at 90 % depth.
The rotational motion exhibited less sensitivity to depth data reduction compared to translational motion, except when no depth data was available, where the SNR dropped below 1. For all other cases, the SNR values consistently initiated with a SNR at approximately 10 across all depth percentages, reaching a peak of 45 at 0.005 rad of simultaneous rotation before subsequently declining.

Feature amount influence

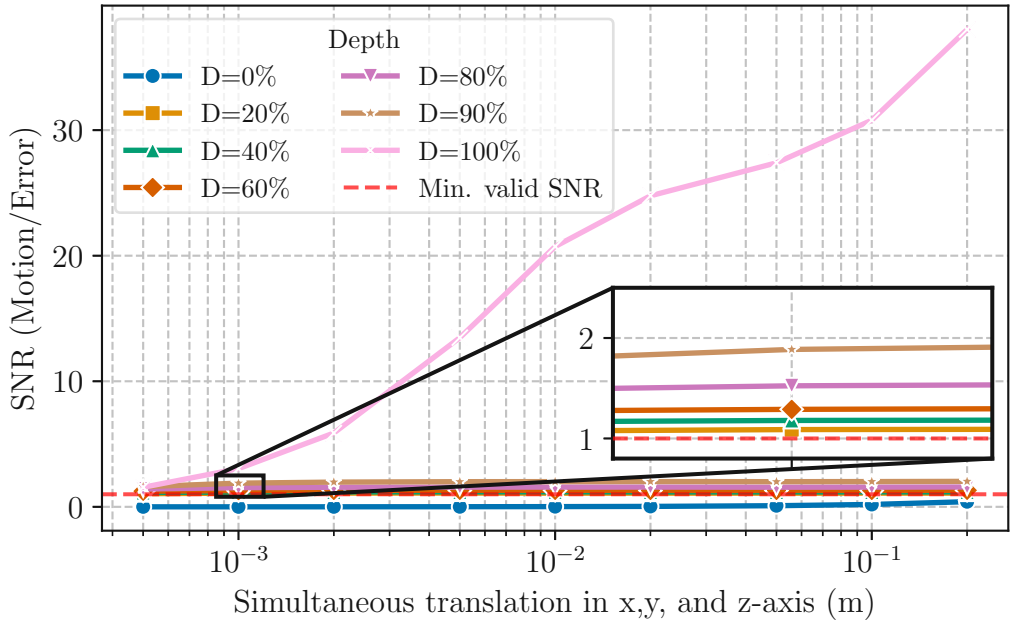
The SNR analysis presented in Figure 4.7 demonstrates how the quantity of initially available and subsequently projected features affects motion estimation accuracy, using a standard deviation noise of one. Both translational and rotational motions exhibit increasing SNR values with larger motion magnitudes. In the translational case, simultaneous motions of 0.5 mm yield SNR values below the minimum valid threshold of 1 across all tested feature quantities. This threshold is only exceeded by all feature quantities when motions surpass 6 mm.
The SNR generally improves with an increased number of projected features used in the estimation process. However, for rotational motion, this trend shows a limitation, with SNR values declining for rotations exceeding 0.01 rad.
Overall, just 45 projected features are enough to detect a rotation of 0.5 mrad with sufficient SNR, while 892 projected features allow for the detection of a 5 mm translation.

Crosstalk rotation and translation

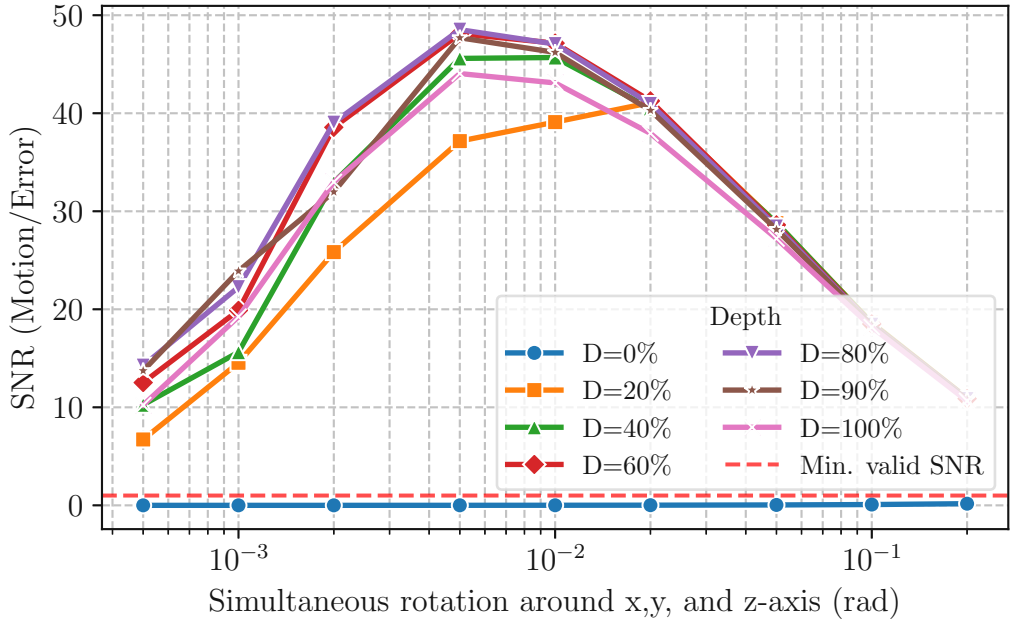
Figure 4.8 illustrates the coupling effects between rotational and translational motions, showing that crosstalk increases proportionally with motion magnitude. At lower motion magnitudes, both rotational and translational crosstalk remain stable at approximately 0.6 mm and 5 mrad, respectively. However, when simultaneous motion reaches around 0.01 m and 0.01 rad, the translational crosstalk error grows linearly, while the rotational crosstalk follows an exponential trend.

4.2.2 Experiment: Iterative Learning Control

The results obtained from the experiments described in the method Section 4.1.5 are now presented and described.

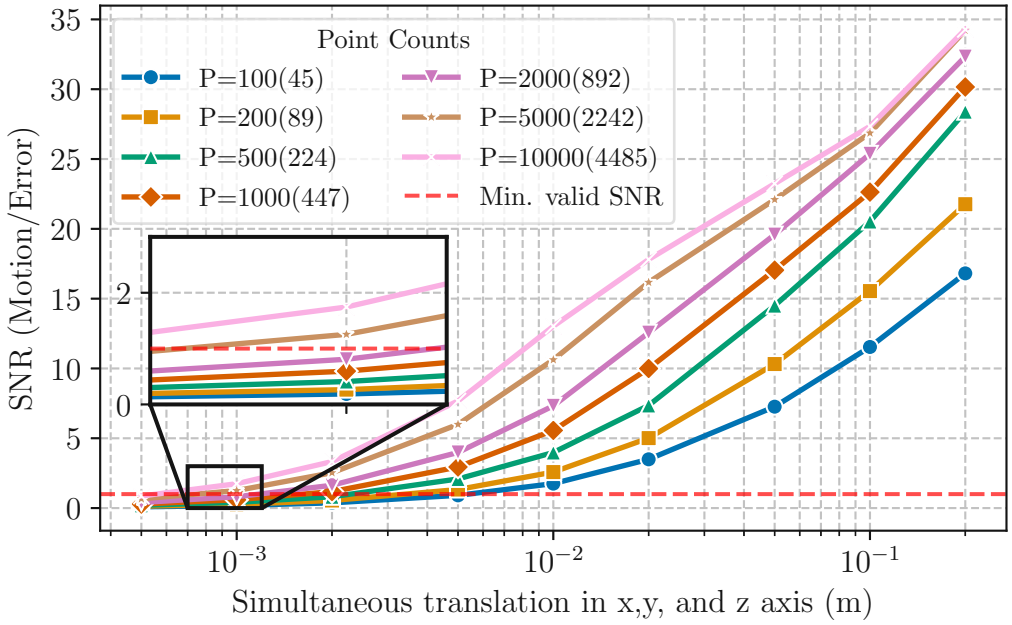


(a)

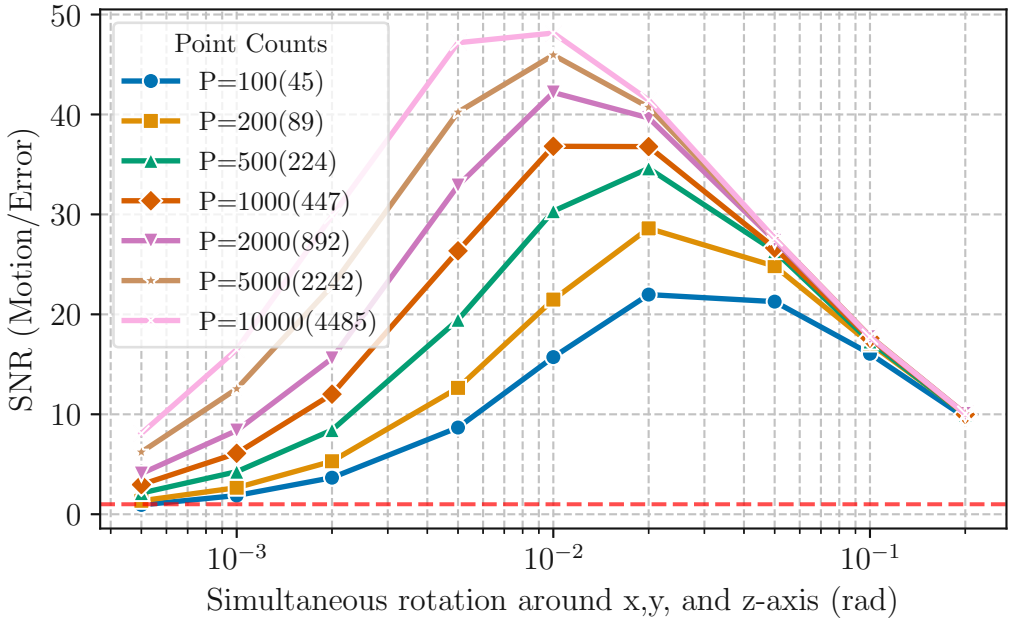


(b)

Figure 4.6: Simulation results showing the influence of the amount of depth information on camera motion estimation with a standard deviation noise of one: (a) simultaneous translation and (b) simultaneous rotation along the x, y, and z axes. The depth ratio indicates the percentage of features with depth information.



(a)

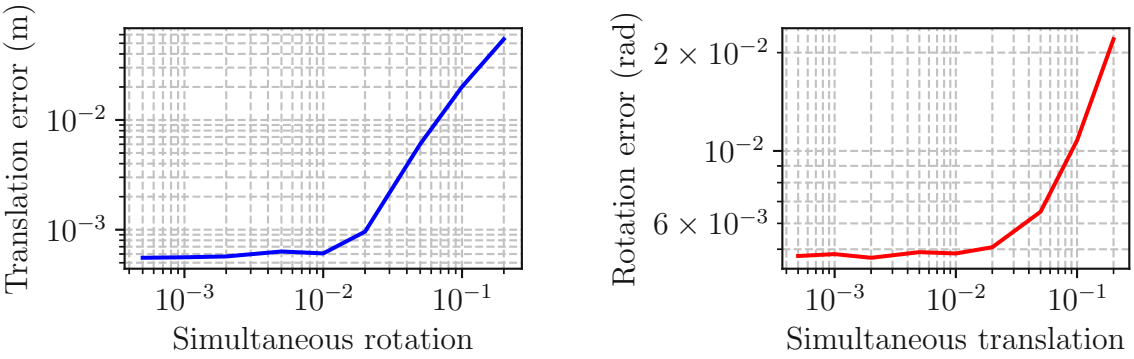


(b)

Figure 4.7: Simulation results showing the influence of the total number of features used for camera motion estimation with a standard deviation noise of 1px: (a) simultaneous translation and (b) simultaneous rotation along the x, y, and z axes. The feature amount P indicates the generated 3D points in the scene, with the points projected onto the frame in brackets.

Baseline Performance Assessment

The baseline performance assessment reveals clear velocity-dependent characteristics in the visual odometry system. Motion estimation accuracy, as detailed in Table 4.5,



(a) (b)

Figure 4.8: Illustration of the effect of crosstalk between rotation and translation. Figure a) shows the translational error in a pure rotational movement. Figure b) shows the rotational error in a pure translational movement.

demonstrates proportional degradation in both positional and rotational precision with increasing trajectory velocities. Across different velocity trials, the positional Absolute Trajectory Error (ATE) rises from 0.308 mm to 0.561 mm, while the rotational ATE increases from 0.578 mrad to 1.084 mrad.

Table 4.5: ATE for varying TCP velocities without induced deviations.

Trial	TCP Velocity ($m\,s^{-1}$)	ATE Position (mm)	ATE Rotation (mrad)
10%	0.0494 ± 0.0014	0.308 ± 0.042	0.578 ± 0.089
20%	0.097 ± 0.008	0.386 ± 0.066	0.737 ± 0.111
30%	0.142 ± 0.018	0.455 ± 0.063	0.788 ± 0.098
40%	0.184 ± 0.033	0.561 ± 0.061	1.084 ± 0.139

Camera Mount Characteristics

The resulting iteration assessment displayed in Figure 4.9 reveals increasing translational and rotational drift in the flexible mount configuration, while the ATE of the rigid reference maintains constant during the experiment. Here, the translational ATE is increased up to 0.07 m, whereas the rotational drift amounts up to 0.10 rad.

Motion Deviation Compensation Assessment

The overall experimental performance of the proposed ILC system is illustrated in Figure 4.10 through ATE measurements across all performed velocity trials (10%-40%). For all velocities, the translational and rotational ATE decreased strongly over the course of the iterations. Higher velocities yield reduced convergence stability, yet all trials achieve final ATE values below 3 mm and 4 mrad for translational and rotational components, respectively.
To quantify this convergence behavior more precisely, a detailed analysis was performed.

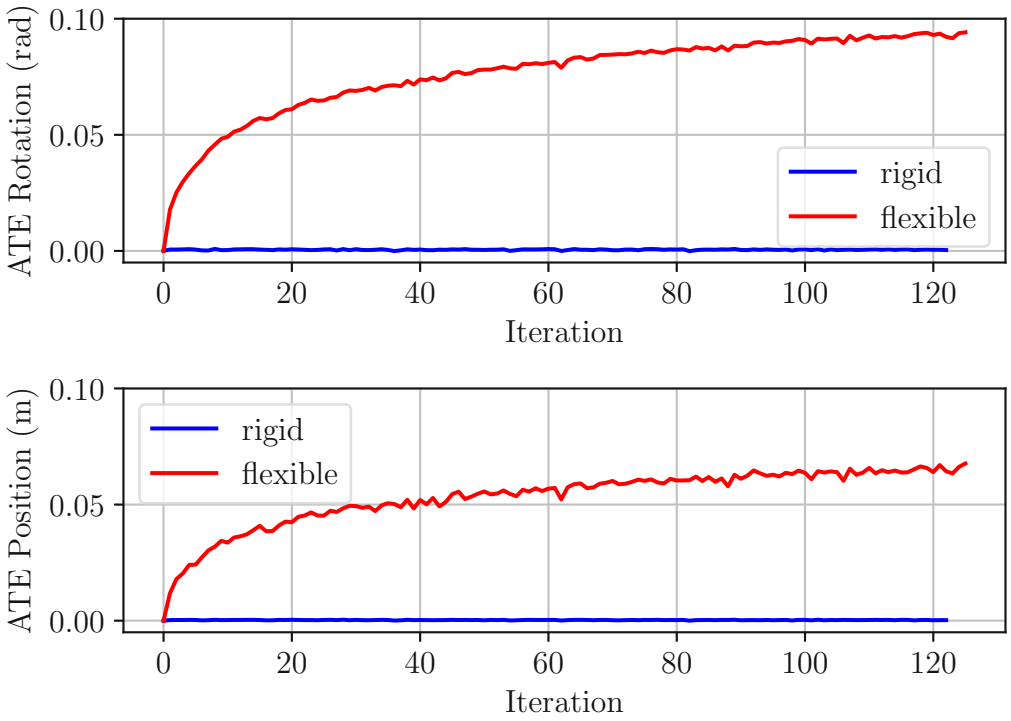


Figure 4.9: Absolute Trajectory Error (ATE) of flexible and rigid camera setup over time without any error mitigation by the ILC.

As shown in Table 4.6, convergence speed exhibits clear velocity dependence. The convergence iteration is determined by analyzing a sliding window of ±5 iterations and identifying the first occurrence where the mean error falls below the threshold. The ILC requires 30-70 iterations for positional convergence, while rotational convergence exhibits longer delays, reaching up to 99 iterations at `40%` velocity. Despite varying convergence rates, the final position accuracy remains consistent with a similar standard deviation of both rotational and translational ATE inside the selected iteration window. The robot joint errors detected and adjusted by the system during the `10%` and `40%`

Table 4.6: ILC convergence analysis: Iterations required to achieve stable mean position and rotation error values (<3 mm, <4 mrad) across velocity trials

Trial	Position		Rotation	
	Iteration	ATE (mm)	Iteration	ATE (mrad)
`10%`	30	2.90 ± 0.26	32	3.80 ± 0.56
`20%`	30	2.92 ± 0.34	33	3.75 ± 0.71
`30%`	46	2.84 ± 0.62	52	3.87 ± 0.43
`40%`	70	2.95 ± 0.30	99	3.98 ± 0.55

trials are presented in Figure 4.11. The data reveals significant variations in initial error magnitudes, with Joint 6 showing the highest initial error at −0.6 rad, while other joints like Joint 1 start with much smaller deviations of 20 mrad. All joints converge to

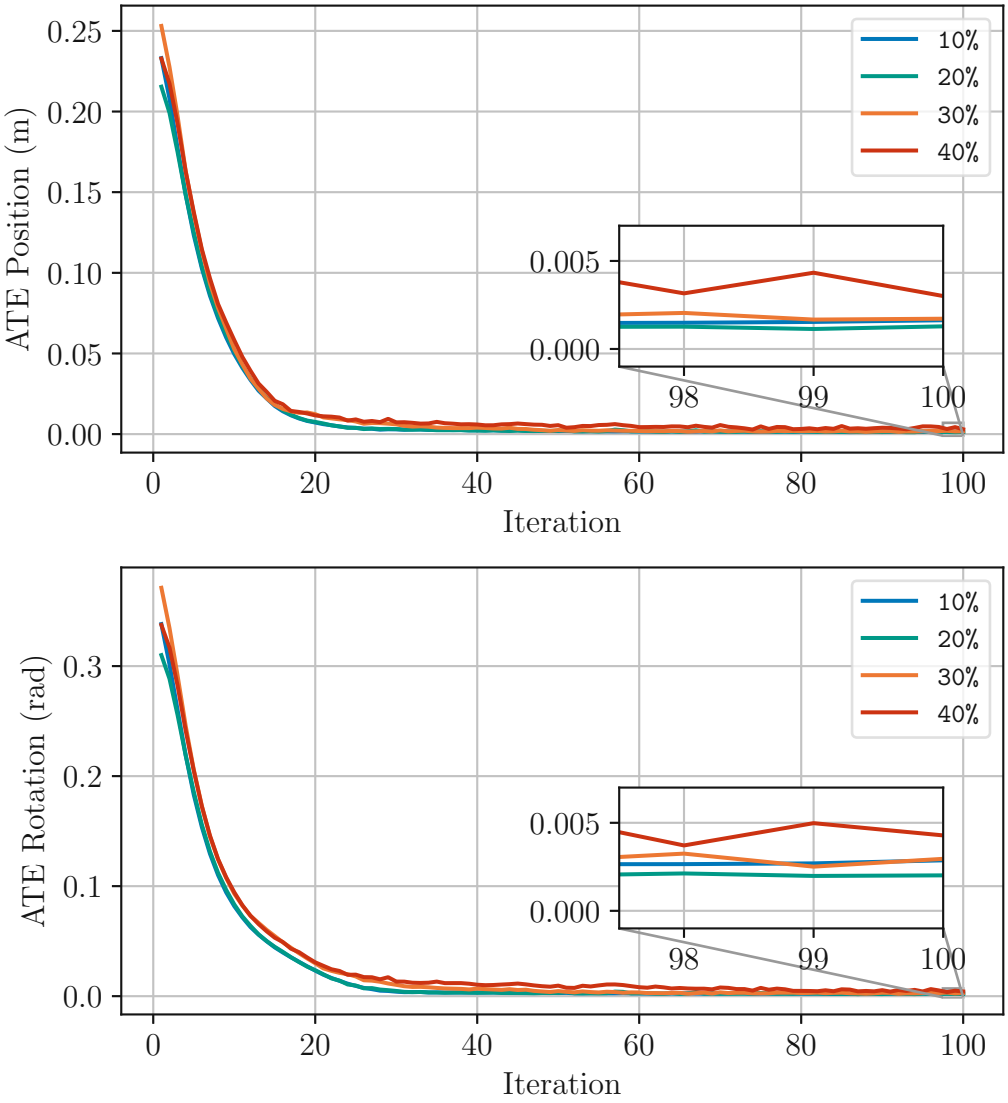


Figure 4.10: Comparison of the proposed vision-based ILC for multiple trials at different velocities (`10%-40%`). For all velocities a learning effect can be observed, where the ILC successfully lowers the ATE over the iterations, converging to a near zero value.

approximately 5 mrad after the 100 iterations, with the `40%` speed trials showing more pronounced oscillations during convergence compared to the `10%` trials.

Dynamic Deviation Compensation Assessment

The dynamic system assessment reveals consistent ATE reduction across all velocity trials in both motion domains, as demonstrated in Figure 4.12. An analysis of isolated high-acceleration waypoints shows increased oscillatory behavior in faster trials (`30%-40%`), though these oscillations weaken over the iterations.
Despite exhibiting lower initial ATE values, the higher velocity trials (`trial_40%` and `trial_30%`) demonstrate a significantly higher translational ATE of over 10 mm at

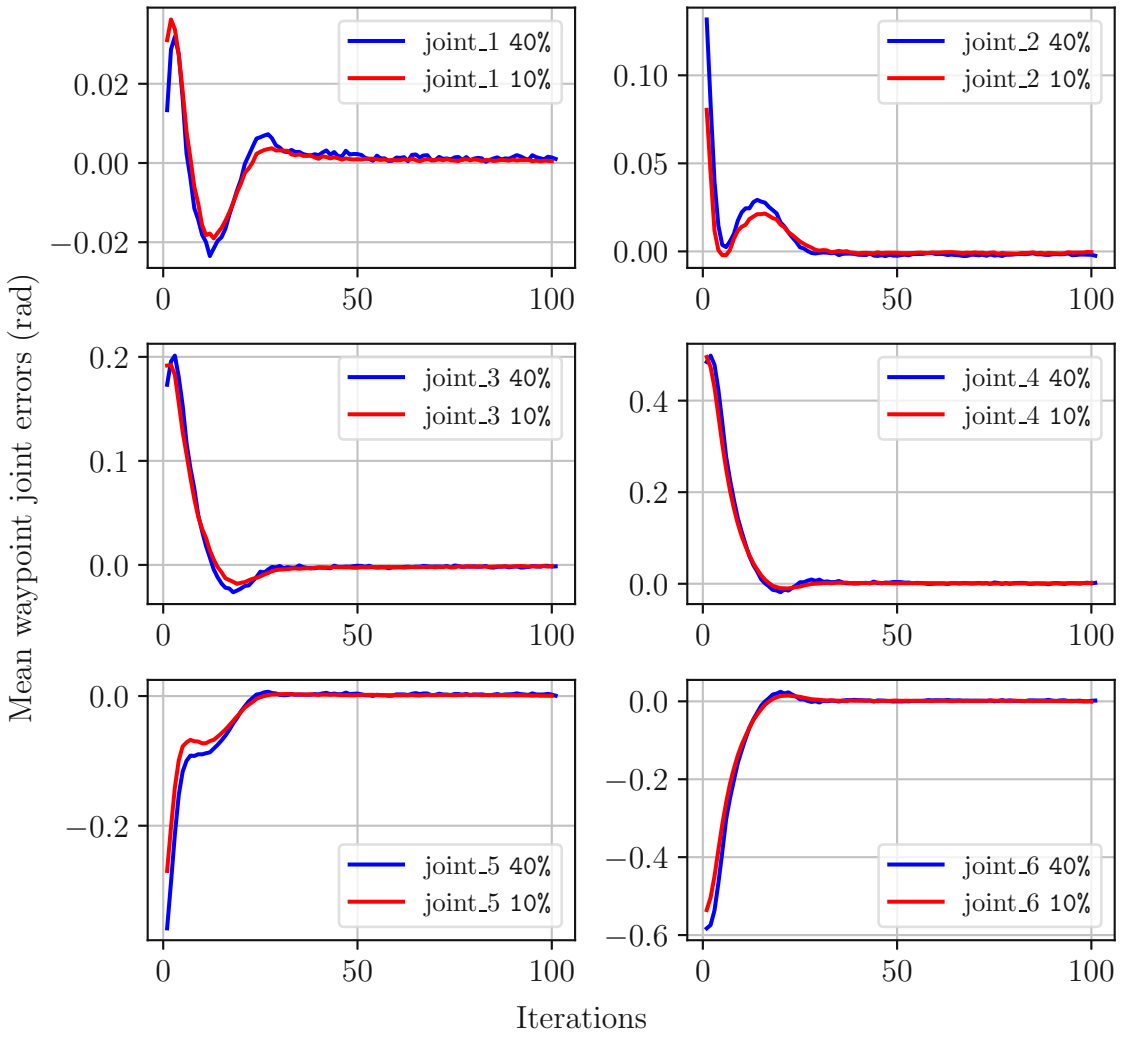


Figure 4.11: Experimental results of the ILC reducing the detected individual mean joint error of the waypoints at different velocity trials (10% and 40%)

iteration 10, compared to approximately 5 mm for lower velocity trials. A similar pattern emerges in the rotational ATE, where higher velocity trials exhibit values approaching 20 mrad at iteration 28, while lower velocity trials maintain values below 4 mrad. At the maximum of 100 iterations, the isolated ATE for higher velocity trials converges to values below 1 mm and 2 mrad, while lower velocity trials achieve even better convergence with values below 1.5 mm and 0.2 mrad.
The mean acceleration data and theoretical end-effector forces in Table 4.7 support this by demonstrating a direct correlation with ATE characteristics, where higher dynamic forces lead to increased error magnitude and variability.
The execution durations for the four trajectories at various velocities are shown in Table 4.8, comparing the initial run with iterations 1, 50, and 100. Higher velocities exhibit more substantial variations in execution time. The most notable increase occurs in the 40% trial, where the 100th iteration requires roughly 20% more time than the initial execution.

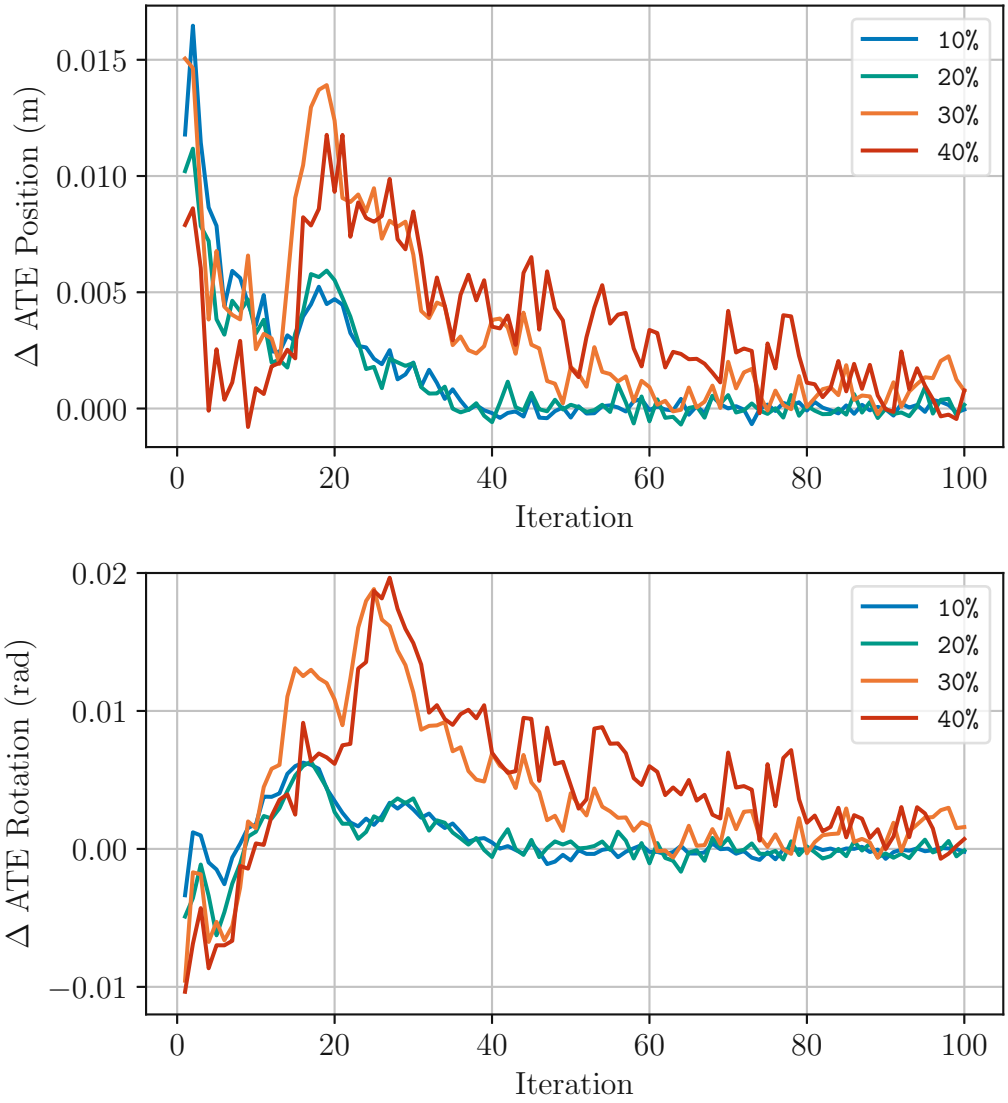


Figure 4.12: System performance of the proposed vision-based ILC multiple velocity trials (10%-40%), with normalized kinematic errors evaluation at high-acceleration waypoints.

Table 4.7: Dynamic ATE for varying TCP acceleration compared to theoretical inertia.

Trial	TCP Acceleration ($m\,s^{-2}$)	Inertia (N)	ATE Position (mm)	ATE Rotation (mrad)
10%	0.05	0.02	0.90 ± 2.17	0.47 ± 0.46
20%	0.31	0.15	0.95 ± 2.23	0.72 ± 0.82
30%	0.68	0.33	1.25 ± 2.48	1.78 ± 1.76
40%	1.2	0.59	1.99 ± 2.23	3.37 ± 2.53

Table 4.8: Trajectory execution times (in seconds) across the ILC iterations for different velocity trials

Trial	Initial	Iteration 1	Iteration 50	Iteration 100
`trial_10%`	12.20	11.55	11.93	11.94
`trial_20%`	6.64	6.26	6.26	6.22
`trial_30%`	4.32	4.32	4.54	4.54
`trial_40%`	3.19	3.19	3.66	3.82

4.3 Discussion

The results presented in Section 4.2 are now discussed and analyzed in more detail. The analysis will consider the effect of the different influences on the SNR in the simulation, as well as the crosstalk between the estimated translation and rotation. Finally, the experiments on the 6-axis robot with the adaptable camera attachment are discussed, and its limitations are described.

4.3.1 Simulation

The simulation of the VO motion estimation system demonstrates robust performance even under challenging conditions such as noise and feature sparsity. The simulated SNR of the system proves sufficient for reliable camera deviation detection, validating its application as an ILC sensor.
Nevertheless, the simulation shows the influence of the hardware, i.e. the camera used, on the quality of motion detection. Thus, the data with different amplitudes of noise (cf. Figure 4.5) illustrate the need for ideal conditions regarding the camera setup and the lighting in order to minimize the influence of the ever-present noise.
The amount of available 3D data shows a connection to the scale ambiguity often mentioned in the literature, as the translational movement, is strongly distorted by a lack of 3D data and quickly falls below the necessary SNR.
The number of projected features, on the other hand, illustrates the need for a high number of features in order to be able to recognize smaller translational movements sufficiently.
The consistent decline in rotational SNR observed across all investigated influences for larger motion magnitudes can be attributed to the limitations of the employed $\hat{\mathbf{R}}^T \cdot \mathbf{R}$ metric, which becomes less suitable for evaluating rotational deviations as their magnitude increases.
Cross-coupling effects between rotational and translational motions remain minimal within the ILC's operational range, as seen in Figure 4.8. The induced crosstalk is consistently smaller than the primary motion and maintains compatibility with convergence criteria across iterations.

4.3.2 Experiment

This subsection discusses the baseline of the experimental setup, the general performance of the ILC, and the dynamic influence of the robot end-effector on the ILC.

ATE Drift Evaluation

In the experimental baseline investigation for the camera attachment, a considerable ATE drift is detected for the flexible setup compared to the rigid setup as seen in Figure 4.9. Over the iterations, both translational and rotational ATE exhibited logarithmic growth from near-zero initial values, reaching peaks exceeding 0.06 m and 0.09 rad respectively.
This drift is due to the constant gravitational force applied on the camera, which, after removing the rigid hull, lowers the attachment further over time. However, the steady convergence of the ATE during the ILC updates shows that this drift impact is nevertheless steadily reduced as seen in Figure 4.10. Only the convergence towards zero is delayed as a result, as the drift effect needs to settle before the ILC can compensate for it.

Vision-based ILC Evaluation

With the results of the experimental results, it can be clearly shown that the vision-based ILC concept can both detect and reduce the camera's motion deviations. As a result, the accuracy of the given trajectory with the flexible setup can be improved by over 98 % after 30 iterations.

While these results are promising, they cannot be generalized to all possible motion paths. Even if the initial trajectory is feasible, the ILC's corrective adjustments might generate waypoints that fall outside the robot's operational capabilities, either exceeding joint limitations or workspace constraints, thus making perfect error compensation unattainable.
Another limitation arises when the corrected trajectory approaches robotic singularities, where certain motions become impossible due to the loss of degrees of freedom.
These limitations could be mitigated through implementation on a robot with seven or more degrees of freedom, where the additional degrees of freedom provide redundancy in motion planning, allowing more flexible trajectory optimization while maintaining the desired end-effector pose, even in near-singular configurations.

Velocity Influence Evaluation

The velocity trials, where the same ILC setup is executed with different end-effector velocities, demonstrate a clear correlation between the dynamic effect on the flexible setup and both error detection and ILC convergence characteristics.
The velocity trials reveal substantial quantitative differences in system accuracy between velocity scenarios. Higher velocity trials demonstrate translational errors 100% larger than lower velocity trials during early convergence (iteration 10), with an even more significant 400% difference in rotational errors at iteration 28. At final convergence, this

performance gap remains evident, with the high-velocity translational and rotational errors being 567% and 900% larger respectively, compared to low-velocity trials.
This difference in ATE indicates, that the VO implementation thus recognizes both the kinematic and the dynamic errors, with a subsequent compensation via the ILC (cf. Figure 4.12). However, the high-velocity trials take longer to converge to the same minimum values. This lower convergence rate could be decreased with further tuning the ILC-parameter, which exceeded the scope of this thesis which focuses on the fundamental validation of the system.
The reasons for the poorer convergence behavior at higher speeds are manifold. On the one hand, the robot's high end-effector velocities introduce additional timing challenges. The precise camera triggering becomes increasingly important since minimal timing variations can result in significant positional discrepancies between the executed iterations and the ground truth recordings. On the other hand, the end-effector experiences the dynamic influence mentioned above, which might also change over the course of the iterations, as the flexibility of the camera mounting is not uniformly the same in each end-effector position.

Initial State Evaluation

The PD-Type ILC's effectiveness extends to correcting initial state errors, a significant practical advantage of the system. This robustness to starting position deviations is particularly valuable for real-world robotic applications, where perfect initial conditions are rarely achievable.

4.3.3 Error Sources and Limitations

In this subsection, potential error sources, as well as inherent limitations of the setup, are discussed.

Environmental Error Influence

As the robot's environment serves as the sole source for motion detection using VO, it is immensely vital for efficient detection. Both the number of features in general and their distribution in 3D space, as well as their illumination, play an important role since, as shown in simulation results (cf. Section 4.3.1), noise and feature versatility play an important role in the accuracy of the measured camera movement.
It is also important that the environment must not change significantly during the individual trajectory execution of the ILC, as any change would be misinterpreted as a movement of the camera. The movement of individual smaller objects would be balanced out by the Random Sample Consensus (RANSAC) filtering of the feature matches, but a change of objects with many features would be potentially fatal.

Camera Limitation

The precision of motion estimation, particularly in the 3D depth reconstruction, depends significantly on the camera's intrinsic parameters obtained during calibration. These

parameters aim to compensate for lens-induced distortions in the camera model, which can affect the accuracy of feature reprojection and affect the overall accuracy of the visual tracking system.
In addition, the camera's frame rate and trigger timing play a crucial role in trajectory quality. Insufficient sampling rates can result in angular, discontinuous motion paths due to limited data points, affecting both the overall smoothness of motion execution and the accuracy of trajectory reconstruction. Higher sampling frequencies allow for more detailed motion capture, leading to smoother, more natural trajectories.

Trajectory Duration

The ILC-updated trajectory may require different joint movements to achieve the desired corrections, potentially increasing or decreasing the path length between waypoints. While the robot's external controller maintains the specified trial speed, these changes in joint displacements result in varying overall trajectory execution times (cf. Table 4.8). This time variation effect could be mitigated through the implementation of a more flexible robot control system that performs independent time parameterization of the trajectory.

4.3.4 Results Summary

The initial simulation results highlight the importance of the environment and camera choice for reliably recognizing camera movement. Factors such as noise, available 3D depth data, and the number of features significantly impact the SNR, which is crucial for meaningful use in the ILC.
The results demonstrate that even under challenging environmental conditions, with a noise standard deviation of 1 pixel and only 20% depth data availability, the SNR remains above the necessary threshold of 1, enabling the detection of small deviations of 0.5 mm and 0.5 mrad. Notably, just 45 projected features are sufficient to detect a rotation of 0.5 mrad with reliable SNR, while 892 projected features allow for the detection of a 5 mm translation, even in these adverse conditions.

The subsequent real-world experiment further confirms the simulation findings. For a given trajectory with two highly dynamic turning points and an additional applied flexibility setup, the deviation is reduced from an ATE of 0.25 m and 0.3 rad to below 3 mm and 5 mrad across different end-effector velocities. Further results show that both the kinematic errors and the non-kinematic errors can be recognized and subsequently compensated.
In summary, the proposed framework is able to enhance the accuracy for predefined tasks in robotic applications, such as repetitive pick-and-place movements, and thus enables high-precision motion.

CHAPTER 5

Conclusion & Outlook

The vision-based Iterative Learning Control (ILC) approaches offer a solution to a major drawback of using collaborative robots (cobots) in industrial settings. Cobots have flexible structures that may cause constant deviations in their trajectories. Vision-based ILC can recognize and compensate for these deviations, thereby increasing the accuracy and enabling the successful execution of a given task.

In the following section, the conclusions of the introduced novel vision-based eye-in-hand ILC approach that relies solely on the environment are presented. Additionally, an outlook on potential future work to further improve the robustness of the trajectory updates and enhance the accuracy of the motion estimation of the proposed system is provided.

5.1 Conclusion

This thesis introduces a novel approach that combines the well-known and extensively studied control concept of ILC with an eye-in-hand camera mounted on an industrial robot. The primary goal is to detect deviations in the robot's end-effector trajectory during repetitive tasks and iteratively improve the execution accuracy through learning from previous iterations. The camera uses only the robot's environment to estimate these deviations, which serve as the sole input for the ILC algorithm to update the robot's trajectory. Deviations introduced by compliant parts of the robot can, therefore, be detected and mitigated using only its environment, eliminating the need for external measurement systems as well as special markers or calibration targets while maintaining high accuracy.
The necessary components and their implementation are discussed in Chapter 3. In summary, the camera captures images during a robot's trajectory execution for two different workflow phases: establishing ground truth and updating the trajectory. For ground truth, the images are saved and used to create a 3D map through triangulation. During the update phase, the images are employed to estimate the camera movement deviation using Visual Odometry (VO).
The VO process relies on an image feature optimization-based algorithm [88], which accommodates features with and without additional depth data. The estimated camera motion, combined with the robot's known kinematic chain, enables the calculation of individual joint errors. These errors subsequently serve as input for the ILC algorithm to update the trajectory in the next iteration.
This process is implemented using a state machine architecture within the Robot Operating System (ROS) framework [78]. ROS facilitates seamless communication between the hardware and software components, ensuring reliable and timely execution of the system.

The system's effectiveness is validated through both simulations and real-world experiments. The simulations demonstrate the robustness of the VO motion estimation system under challenging conditions, including varying levels of camera noise, different densities of available feature points, and varying quality of 3D depth information, validating its application as an ILC error input.
The experimental setup utilizes a multipurpose, both flexible and rigid camera-to-robot setup to easily perform the ground truth and update trajectory executions. The ILC algorithm demonstrated high learning capability, reducing the initially applied Absolute Trajectory Error (ATE) of 0.25 m and 0.3 rad to below 3 mm and 4 mrad respectively, representing an improvement in trajectory accuracy of over 98.5% compared to the initial flexible robot execution.
Further investigations demonstrated the system's robustness across varying end-effector velocities. The VO system successfully detected, and the ILC effectively compensated for, dynamic effects caused by end-effector inertia. While trajectories with higher velocities required additional learning iterations, the system consistently achieved a similar high accuracy levels, confirming capability to detect and correct both kinematic and non-kinematic errors.

The results presented in this work demonstrate that a vision-based ILC approach

relying only on environmental features is suitable for trajectory error compensation in repetitive industrial robot tasks. By efficiently compensating for undesirable flexibilities typically present in cobots, the system achieves high accuracy. Using only a single camera at the robot's end-effector, it operates without requiring external measurement systems, artificial markers, or calibration targets. Thus, this work lays the foundation for developing practical vision-based control solutions that address flexibility challenges in industrial collaborative robotics.

5.2 Outlook

Although this work successfully demonstrated its capability to detect and compensate kinematic and non-kinematic errors, the experimental setup employed an exaggerated flexible camera attachment to validate the approach. For real industrial applications, where the flexibility effects of a robot system are less apparent, the detection accuracy of the VO system needs to be improved to capture and compensate for these smaller deviations.
One major technical limitation of the current setup stems from the camera's frame rate and trigger timing. The maximum end-effector velocity is constrained, as sufficient image sampling during the trajectory must be maintained for reliable VO estimation. This is particularly critical during highly dynamic portions of the trajectory, where higher sampling rates are needed to reliably detect and compensate for rapid movements and accelerations. Future work should investigate cameras with higher frame rate and improved trigger timing to increase both the motion estimation accuracy and the maximum achievable velocity while maintaining the required image capture density.

Beyond the hardware limitations, the ground truth generation in the current approach requires specific test conditions that are not practical in industrial settings. One alternative method would be to use a slow reference trajectory execution with minimal dynamic effects for ground truth generation, followed by high-speed ILC executions where these effects can be learned and compensated. Another approach could involve performing executions with and without end-effector payload to identify and compensate for flexibility-induced deviations.

Bibliography

[1] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, "Intelligent manufacturing in the context of industry 4.0: A review," *Engineering*, vol. 3, no. 5, pp. 616–630, 2017.

[2] J.-P. Merlet, "Optimal design of robots," in *Robotics: Science and systems*, 2005.

[3] Blue Danube Robotics GmbH, *Airskin fenceless industrial robot*, https://www.airskin.io/ [Accessed: 24.01.2025], 2025.

[4] B. Fu and G. Cai, "Design and calibration of a joint torque sensor for robot compliance control," *IEEE Sensors Journal*, vol. 21, no. 19, pp. 21 378–21 389, 2021.

[5] R.-J. Halme, M. Lanz, J. Kämäräinen, R. Pieters, J. Latokartano, and A. Hietanen, "Review of vision-based safety systems for human-robot collaboration," *Procedia Cirp*, vol. 72, pp. 111–116, 2018.

[6] F. Vicentini, "Collaborative robotics: A survey," *Journal of Mechanical Design*, vol. 143, no. 4, p. 040 802, 2021.

[7] T. Varhegyi, M. Melik-Merkumians, M. Steinegger, G. Halmetschlager-Funek, and G. Schitter, "A Visual Servoing Approach for a Six Degrees-of-Freedom Industrial Robot by RGB-D Sensing," *Autom. Robot*, 2017.

[8] *ISO 8373:2021(en) Robotics — Vocabulary*, Geneva, Switzerland: ISO, 2021.

[9] M. Ben-Ari and F. Mondada, *Elements of robotics*. Springer Nature, 2017.

[10] P. Dario, E. Guglielmelli, B. Allotta, and M. C. Carrozza, "Robotics for medical applications," *IEEE Robotics & Automation Magazine*, vol. 3, no. 3, pp. 44–56, 1996.

[11] J. Forlizzi and C. DiSalvo, "Service robots in the domestic environment: A study of the roomba vacuum in the home," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, 2006, pp. 258–265.

[12] C. Badue *et al.*, "Self-driving cars: A survey," *Expert systems with applications*, vol. 165, p. 113 816, 2021.

[13] M. R. Pedersen *et al.*, "Robot skills for manufacturing: From concept to industrial deployment," *Robotics and Computer-Integrated Manufacturing*, vol. 37, pp. 282–291, 2016.

[14] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control* (Advanced Textbooks in Control and Signal Processing). Springer, 2009.

[15] S. Kucuk and Z. Bingul, *Robot kinematics: Forward and inverse kinematics*. INTECH Open Access Publisher London, UK, 2006.

[16] R. P. Paul and B. Shimano, "Kinematic control equations for simple manipulators," in *1978 IEEE Conference on Decision and Control including the 17th Symposium on Adaptive Processes*, IEEE, 1979, pp. 1398–1406.

[17] J. B. Kuipers, *Quaternions and rotation sequences: a primer with applications to orbits, aerospace, and virtual reality*. Princeton university press, 1999.

[18] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," *Journal of Applied Mechanics*, vol. 22, no. 2, pp. 215–221, 1955.

[19] R. W. Brockett, "Robotic manipulators and the product of exponentials formula," in *Mathematical Theory of Networks and Systems: Proceedings of the MTNS-83 International Symposium Beer Sheva, Israel, June 20–24, 1983*, Springer, 2005, pp. 120–129.

[20] K. M. Lynch and F. C. Park, *Modern robotics*. Cambridge University Press, 2017.

[21] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir, "Inverse kinematics techniques in computer graphics: A survey," in *Computer graphics forum*, Wiley Online Library, vol. 37, 2018, pp. 35–58.

[22] A. S. Deo and I. D. Walker, "Overview of damped least-squares methods for inverse kinematics of robot manipulators," *Journal of Intelligent and Robotic Systems*, vol. 14, pp. 43–68, 1995.

[23] M. Shimizu, H. Kakuya, W.-K. Yoon, K. Kitagaki, and K. Kosuge, "Analytical inverse kinematic computation for 7-dof redundant manipulators with joint limits and its application to redundancy resolution," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1131–1142, 2008.

[24] Y. He and S. Liu, "Analytical inverse kinematics for franka emika panda–a geometrical solver for 7-dof manipulators with unconventional design," in *2021 9th International Conference on Control, Mechatronics and Automation (ICCMA)*, IEEE, 2021, pp. 194–199.

[25] S. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Transactions in Robotics and Automation*, vol. 17, May 2004.

[26] C. W. Wampler, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 93–101, 1986.

[27] W. H. Press, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[28] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 1999.

[29] D. G. Luenberger, Y. Ye, *et al.*, *Linear and nonlinear programming*. Springer, 1984, vol. 2.

[30] L.-C. Wang and C.-C. Chen, "A combined optimization method for solving the inverse kinematics problems of mechanical manipulators," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 4, pp. 489–499, 1991.

[31] A. Aristidou and J. Lasenby, "FABRIK: A fast, iterative solver for the Inverse Kinematics problem," *Graphical Models*, vol. 73, no. 5, pp. 243–260, 2011.

[32] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *2011 IEEE international conference on robotics and automation*, IEEE, 2011, pp. 1478–1483.

[33] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[34] D. Bristow, M. Tharayil, and A. Alleyne, "A survey of iterative learning control," *IEEE Control Systems Magazine*, vol. 26, no. 3, pp. 96–114, 2006.

[35] S. Arimoto, S. Kawamura, and F. Miyazaki, "Bettering operation of robots by learning," *Journal of Robotic systems*, vol. 1, no. 2, pp. 123–140, 1984.

[36] M. Phan, J.-N. Juang, and R. W. Longman, "On markov parameters in system identification," Tech. Rep., 1991.

[37] T. Sogo, K. Kinoshita, and N. Adachi, "Iterative learning control using adjoint systems for nonlinear non-minimum phase systems," in *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187)*, IEEE, vol. 4, 2000, pp. 3445–3446.

[38] P. Janssens, G. Pipeleers, and J. Swevers, "A data-driven constrained norm-optimal iterative learning control framework for LTI systems," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 2, pp. 546–551, 2012.

[39] C. Liu, M. Wang, X. Li, and S. Ratchev, "Feedforward enhancement through iterative learning control for robotic manipulator," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2021, pp. 1067–1072.

[40] J. Wallén, M. Norrlöf, and S. Gunnarsson, "Arm-side evaluation of ILC applied to a six-degrees-of-freedom industrial robot," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 13 450–13 455, 2008.

[41] Y. M. Zhao, Y. Lin, F. Xi, and S. Guo, "Calibration-based iterative learning control for path tracking of industrial robots," *IEEE Transactions on industrial electronics*, vol. 62, no. 5, pp. 2921–2929, 2014.

[42] B. Jia, S. Liu, and Y. Liu, "Visual trajectory tracking of industrial manipulator with iterative learning control," *Industrial Robot: An International Journal*, vol. 42, no. 1, pp. 54–63, 2015.

[43] J. Wu, Z. Jin, A. Liu, l. yu li, and F. Yang, "A survey of learning-based control of robotic visual servoing systems," *Journal of the Franklin Institute*, vol. 359, Nov. 2021.

[44] G. Heinzinger, D. Fenwick, B. Paden, and F. Miyazaki, "Stability of learning control with disturbances and uncertain initial conditions," *IEEE Transactions on Automatic Control*, vol. 37, no. 1, pp. 110–114, 1992.

[45] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[46] J. Wang, F. Shi, J. Zhang, and Y. Liu, "A new calibration model of camera lens distortion," *Pattern recognition*, vol. 41, no. 2, pp. 607–615, 2008.

[47] R. Y. Tsai, "Efficient and accurate camera calibration technique for 3d machine vision.," in *IEEE conference on computer vision and pattern recognition*, 1985.

[48] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[49] Y. Jiang, S. Karpf, and B. Jalali, "Time-stretch LiDAR as a spectrally scanned time-of-flight ranging camera," *Nature photonics*, vol. 14, no. 1, pp. 14–18, 2020.

[50] A. Mertan, D. J. Duff, and G. Unal, "Single image depth estimation: An overview," *Digital Signal Processing*, vol. 123, p. 103 441, 2022.

[51] A. O' Riordan, T. Newe, D. Toal, and G. Dooly, "Stereo vision sensing: Review of existing systems," Dec. 2018.

[52] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4104–4113.

[53] D. Nister, "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[54] R. I. Hartley and P. Sturm, "Triangulation," *Computer vision and image understanding*, vol. 68, no. 2, pp. 146–157, 1997.

[55] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of applied mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[56] F. Fraundorfer and D. Scaramuzza, "Visual odometry: Part II: Matching, robustness, optimization, and applications," *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 78–90, 2012.

[57] Irani, Rousso, and Peleg, "Recovery of ego-motion using image stabilization," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 1994, pp. 454–460.

[58] A. Howard, "Real-time stereo visual odometry for autonomous ground vehicles," in *2008 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, 2008, pp. 3946–3952.

[59] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Proceedings 2007 IEEE international conference on robotics and automation*, IEEE, 2007, pp. 3565–3572.

[60] M. Li and A. I. Mourikis, "High-precision, consistent EKF-based visual-inertial odometry," *The International Journal of Robotics Research*, vol. 32, no. 6, pp. 690–711, 2013.

[61] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation," in *Robotics: Science and Systems XI*, 2015.

[62] H. Zhan, C. S. Weerasekera, J.-W. Bian, and I. Reid, "Visual odometry revisited: What should be learnt?" In *2020 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2020, pp. 4203–4210.

[63] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[64] D. G. Lowe, "Object recognition from local scale-invariant features," in *International Conference on Computer Vision*, vol. 7, 1999, pp. 1150–1157.

[65] H. Bay, T. Tuytelaars, and L. V. Gool, "SURF: Speeded up robust features," in *European Conference on Computer Vision*, Springer, 2006, pp. 404–417.

[66] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *International Conference on Computer Vision*, 2011, pp. 2564–2571.

[67] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, Springer, 2006, pp. 430–443.

[68] C. Tomasi and T. Kanade, "Detection and tracking of point," *Int J Comput Vis*, vol. 9, no. 137-154, p. 3, 1991.

[69] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration.," *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[70] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[71] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91–110, 2004.

[72] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, no. 5828, pp. 133–135, 1981.

[73] J. Weng, N. Ahuja, and T. S. Huang, "Optimal motion and structure estimation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 15, no. 9, pp. 864–884, 1993.

[74] Z. Zhang and D. Scaramuzza, "A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 7244–7251.

[75] R. M. Murray, Z. Li, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[76] R. Kümmerle *et al.*, "On measuring the accuracy of slam algorithms," *Autonomous Robots*, vol. 27, pp. 387–407, 2009.

[77] *ROS/Introduction - ROS Wiki*, [Online; accessed 17. Jul. 2024], Jul. 2024.

[78] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[79] H. R. Kam, S.-H. Lee, T. Park, and C.-H. Kim, "Rviz: a toolkit for real domain data visualization," *Telecommunication Systems*, vol. 60, no. 2, pp. 337–345, 2015.

[80] *Bags - ROS Wiki*, [Online; accessed 17. Jul. 2024], Jul. 2024.

[81] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[82] S. Agarwal, K. Mierle, and T. C. S. Team, *Ceres Solver*, version 2.2, Oct. 2023.

[83] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: A moveit! case study," *arXiv preprint arXiv:1404.3785*, 2014.

[84] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec. 2012, https://ompl.kavrakilab.org.

[85] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012, pp. 3859–3866.

[86] H. Bruyninckx, "Open robot control software: The orocos project," in *Proceedings 2001 ICRA. IEEE international conference on robotics and automation (Cat. No. 01CH37164)*, IEEE, vol. 3, 2001, pp. 2523–2528.

[87] Photonfocus AG, "Double Rate Camera DR1-D2048x1088-192-G2," 2024.

[88] J. Zhang, M. Kaess, and S. Singh, "Real-time depth enhanced monocular odometry," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014, pp. 4973–4980.

[89] ABB Asea Brown Boveri Ltd, *Irb 120 industrieroboter*, Accessed: 30-01-2025.

Eigenständigkeitserklärung

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct, insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

27.02.2025

Datum

Niklas Paternoster