Two-Dimensional Pose Estimation of Industrial Robotic Arms in Highly Dynamic Collaborative Environments

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Thomas Gulde
aus Geislingen

Tübingen
2022

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	04.04.2023
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr.-Ing Cristóbal Curio
2. Berichterstatter:	Prof. Dr. rer. nat. Andreas Schilling

Abstract

In modern collaborative production environments where industrial robots and humans are supposed to work hand in hand, it is mandatory to observe the robot's workspace at all times. Such observation is even more crucial when the robot's main position is also dynamic e.g. because the system is mounted on a movable platform. As current solutions like physically secured areas in which a robot can perform actions potentially dangerous for humans, become unfeasible in such scenarios, novel, more dynamic, and situation-aware safety solutions need to be developed and deployed.

This thesis mainly contributes to the bigger picture of such a collaborative scenario by presenting a data-driven convolutional neural network-based approach to estimate the two-dimensional kinematic-chain configuration of industrial robot-arms within raw camera images. This thesis also provides the information needed to generate and organize the mandatory data basis and presents frameworks that were used to realize all involved subsystems. The robot-arm's extracted kinematic-chain can also be used to estimate the extrinsic camera parameters relative to the robot's three-dimensional origin. Further a tracking system, based on a two-dimensional kinematic chain descriptor is presented to allow for an accumulation of a proper movement history which enables the prediction of future target positions within the given image plane. The combination of the extracted robot's pose with a simultaneous human pose estimation system delivers a consistent data flow that can be used in higher-level applications.

This thesis also provides a detailed evaluation of all involved subsystems and provides a broad overview of their particular performance, based on novel generated, semi-automatically annotated, real datasets.

“Those people who think they know everything are a great annoyance to those of us who do.”

Isaac Asimov (1920-1992)

Publications

First author publications of this thesis

Parts or ideas of this thesis have been previously published in the following articles. This papers will be referred in this thesis by their roman numerals (eg. [I, II]).

I **Gulde, T.**, Ludl, D. and Curio, C., 2018, August. RoPose: CNN-based 2D pose estimation of industrial robots. In 2018 IEEE 14th International Conference on Automation Science and Engineering (pp. 463-470). IEEE. © 2018 IEEE. [1]

II **Gulde, T.**, Ludl, D., Andrejtschik, J., Thalji, S. and Curio, C., 2019, May. RoPose-Real: real world dataset acquisition for data-driven industrial robot arm pose estimation. In 2019 International Conference on Robotics and Automation (ICRA) (pp. 4389-4395). IEEE. © 2019 IEEE. [2]

Contributions of the Author to the publications

The author of this thesis contributed to the respective publications as follows

I Idea, concept, majority of implementation, data generation and manuscript.

II Idea, concept, majority of implementation, data generation and manuscript.

Nr.	Accepted publication yes/no	List of authors	Position of candidate in list of authors	Scientific ideas by the candidate [%]	Data generation by the candidate [%]	Analysis and Interpretation by the candidate [%]	Paper writing done by the candidate [%]
1 [1]	yes	T. Gulde, D. Ludl, C. Curio	1	90	90	90	100
2 [2]	yes	T. Gulde, D. Ludl, J. Andrejtschik, S. Thalji, C. Curio	1	90	75	90	100

Overview of the authors' contribution in published main papers with co-authors.

Further publications related to this thesis

In addition to the work mentioned in this thesis, some more projects and collaborations resulted in the following publications. They may get referenced like standard papers in the main text.

A **Gulde, T.**, Kärcher, S. and Curio, C., 2016, October. Vision-based slam navigation for vibro-tactile human-centered indoor guidance. In European Conference on Computer Vision (pp. 343-359). Springer [3]

B Ludl, D., **Gulde, T.** and Curio, C., 2020. Enhancing Data-Driven Algorithms for Human Pose Estimation and Action Recognition Through Simulation. IEEE Transactions on Intelligent Transportation Systems, 21(9), pp.3990-3999. [4]

C Ludl, D., **Gulde, T.**, Thalji, S. and Curio, C., 2018, November. Using simulation to improve human pose estimation for corner cases. In 2018 21st International Conference on Intelligent Transportation Systems (pp. 3575-3582). IEEE. [5]

D Ludl, D., **Gulde, T.**. and Curio, C., 2019, October. Simple yet efficient real-time pose-based action recognition. In 2019 IEEE Intelligent Transportation Systems Conference (ITSC) (pp. 581-588). IEEE. [6]

E Baulig, G., **Gulde, T.** and Curio, C., 2018. Adapting egocentric visual hand pose estimation towards a robot-controlled exoskeleton. In Proceedings of the European Conference on Computer Vision (ECCV) Workshops. [7]

F Essich, M., Ludl, D., **Gulde, T.** and Curio, C., 2019, September. Learning to translate between real world and simulated 3D sensors while transferring task models. In 2019 International Conference on 3D Vision (3DV) (pp. 681-689). IEEE. [8]

Contributions of the Author to the publications

The author of this thesis contributed to the respective publications as follows

A Concept, implementation and manuscript.

B Assistance in data generation.

C Assistance in data generation.

D Assistance in data generation.

E Idea and main concept.

F Parts of the perception concept.

Open Source Contributions

Projects directly connected to this thesis

The following open source projects mainly emerged from the work presented in this thesis.

i **ropose** - https://github.com/guthom/ropose
This Python module contains the source code to train, use and test the CNN-based *RoPose* model to estimate the 2D joint position of an industrial robot-arm on common RGB camera images.
Author: T. Gulde

ii **ropose_datagrabber** - https://github.com/guthom/ropose_datagrabber
This C++ ROS-Package contains the datagrabber to independently generate datasets to train the *RoPose* [I, II] system with custom datasets. It was developed to generate only robot-arm specific datasets but evolved in a multi purpose dataset generator. It can now be used to generate ROS-independent datasets based on a variant of visual perception systems (e.g. RGB-Images, PointClouds, depth-maps) with labels based on the transformation information available in ROS's transformation system.
Authors: T. Gulde, J. Andrejtschik

iii **ropose_dataset_tools** - https://github.com/guthom/ropose_dataset_tools
This Python package contains the dataset-tools used to read and organize the datasets created by the *ropose_datagrabber*.
Author: T. Gulde

iv **ropose_greenscreener** - https://github.com/guthom/ropose_greenscreener
This Python package can be used to manipulate the simulated *RoPose*-datasets based on a chroma key solution and the virtual greenscreen used in the simulated datasets.
Author: T. Gulde

v **kinematic_tracker** - https://github.com/guthom/kinematic_tracker
This Python package contains the classes and tools used to organize, track and visualize pose models of industrial robot-arms based on their respective kinematic chains.
Author: T. Gulde

vi **tag_referencer** - https://github.com/guthom/tag_referencer
This C++ ROS-package is used to automatically detect standard QR-Codes or robotic April-Tags [9, 10] within a 2D camera image. The resulting tag information can also be fused with the 3D point cloud information (if available) to stream the estimated 3D pose of the recognized tags directly to the ROS transformation system. The resulting transformations can then be used for a variety of applications just as extrinsic camera calibration, pose referencing or pose based dataset generation.
Author: T. Gulde

vii **kollrobot_controller** - https://github.com/guthom/kollrobot_controller
This C++ ROS-package wraps a *MoveIt!*- controller [11] for the special purposes of the *Kollro 4.0* project (see Chapter 1). The package also has been used to trigger the path planning process and move to random poses of the employed robot-arm to collect pose datasets for *RoPose* and also offers a node to perform extrinsic calibrations based on *AprilTags* [9, 10].
Author: T. Gulde

Contributions of the Author

The author of this thesis contributed to the respective open source projects as follows:

i Sole author of the source code.

ii Main author of the source code, confer to git-commits for specific author information.

iii Sole author of the source code.

iv Sole author of the source code.

v Sole author of the source code.

vi Sole author of the source code.

vii Main author of the source code, confer to git-commits for specific author information.

Further projects related to this thesis

In addition to the work directly related to this thesis, the following open source projects, tools or packages emerged from side projects and other collaborations.

a **labstreaminglayer_ros** - https://github.com/guthom/labstreaminglayer_ros
A Python based ROS-package to directly exchange streams and messages between ROS and Lab Streaming Layer (LSL) [12] which is widely used in neuroscience application.
Authors: T. Gulde, M. Nann, M. Essich

b **guthoms_helpers** - https://github.com/guthom/guthoms_helpers
A Python-package that contains common classes used in various projects. Created to enforce consistency in file handling, data processing and especially transformation handling within multiple projects.
Authors: T. Gulde, M. Essich

c **custom_parameters** - https://github.com/guthom/custom_parameters
A ROS-package that wraps the ROS parameter system in a convenient template based C++ library. However, this custom packaged could became obsolete with the rise of dynamic reconfigure [13].
Author: T. Gulde

d **magic_box** - https://github.com/guthom/magic_box
This Python based ROS-package was used as a driver node for a smart container prototype to enable local references for dynamic pick and place task with industrial robots. The box has been equipped with an internal ink-display to show *AprilTags* [9, 10] which can be changed via the implemented ROS-services.
Authors: T. Gulde, V. V. Nair, J. Schuhmacher

Contributions of the Author

The author of this thesis contributed to the respective open source projects as follows:

a Main author of the source code, confer to git-commits for specific author information.

b Main author of the source code, confer to git-commits for specific author information.

c Sole author of the source code.

d Main author of the source code, confer to git-commits for specific author information.

Contents

1 Introduction

"Yes, excessive automation at Tesla was a mistake. To be precise, my mistake. Humans are underrated."

Elon Musk, 2018 - [14]

1.1 Motivation

The rapid evolution of production and manufacturing applications in the last decades poses new challenges for almost every involved technology sector [15]. Manufacturing and all related subtasks transform in more and more agile processes that even exceed the traditional understanding of lean- [16, 17] and just-in-time manufacturing [18]. When novel customer requirements like customization join modern management and engineering needs (e.g. advanced performance monitoring, analytics, and rapid prototyping) even more digital and physical systems take over important roles within the creation process. [19]

As human individuals are an essential part of these processes and tasks, their collaboration and interaction with the rising number of involved systems have also moved into the design-focus of modern manufacturing [20]. However, humans add an additional, highly dynamic and not deterministic component to the ensemble and thus require extra safety and security efforts [21]. Especially when it comes to full collaborative environments, where humans and e.g. industrial robots have to act dynamically hand in hand to solve complex tasks, this introduced dynamic has to be transferred into almost all subsystems. Therefore, it became more and more important to reliably perceive, supervise, and monitor these environments, especially when the final areas of collaborative zones can not be preassigned. In order to advance applications, smart sensory systems just as vision sensors are needed. These sensors need to be capable to autonomously recognize and localize humans and the involved robotic system within the observed space, ideally without the need for additional periphery devices.

Because the authors research group faced such problems in various projects involving collaborative tugger trains with industrial robot manipulators [1] (see e.g. [23] for more details) and also hand exoskeletons [2], The research presented in this thesis focuses on the Computer Vision (CV) based observation of dynamic articulated targets in robotic

[1]Project: KollRo 4.0 [22]

[2]Project: KONSENS-NHE [24]

environments. Some fundamental parts of this thesis are based on disruptive machine learning technologies like deep learning and especially Convolutional Neural Networks (CNNs). As these technologies had a huge impact on the field of CV [25] and robotics (cf. Section 1.3), it is obvious that such technologies enable new prospects and have great potential to also improve collaborative applications.

1.2 Collaborative Robotics and its Challenges

Collaboration is a widely used word to describe overlapping or joint activities between companies, organizations, research groups, technical systems, and human individuals. However, in this work, the term always implies the interaction of two or more systems (e.g. humans, robots) within a targeted value creation process aiming to successfully complete a given task. This work will primarily concentrate on collaborative robotics and manufacturing systems where humans and industrial robot arms directly share a physical and predominantly not deterministic workspace (Human-Robot Collaboration (HRC)). Figure 1.1 shows a simplified example of such an environment.

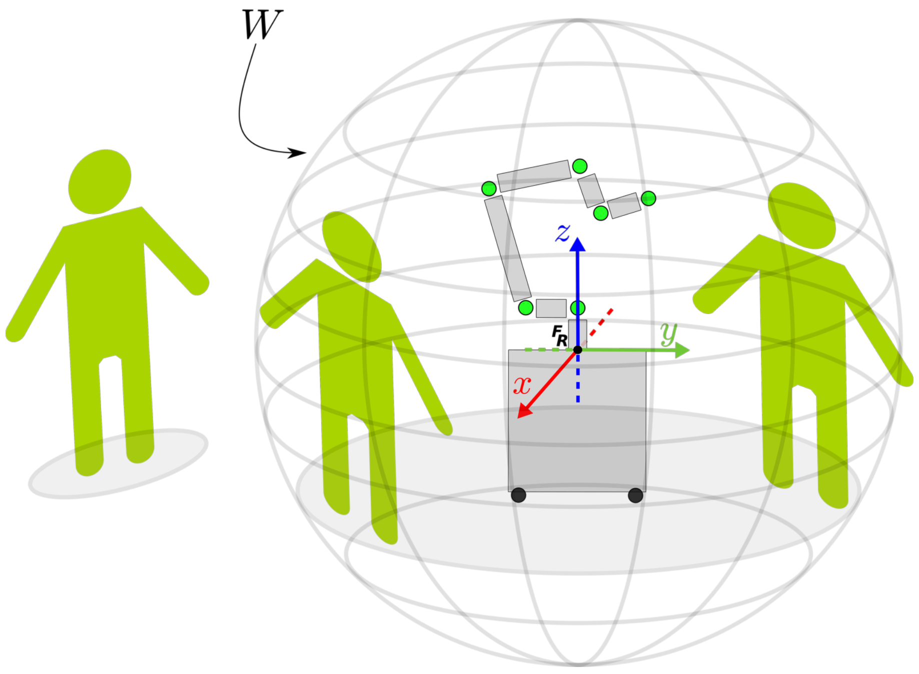


Figure 1.1: Shared workspace between a mobile industrial robot-arm and humans in HRC scenarios. W: idealized Three-Dimensional (3D) workspace of a robot, F_R: Robot coordinate system originates at the base of the robot usually also defines the origin of the robot's workspace.

The workspace W of an industrial robot describes the space a manipulator can reach and is defined by the arm's physical constraints and the given kinematics. The workspace is therefore typically not as ideal as shown above. The so-called Configuration Space (C-Space) describes all configurations (~poses) a robot can engage and takes obstacles and other non reachable areas into account [26]. When the main actuator is also mounted to a moving platform, the workspace and thus also the C-space of the manipulator can be extended to almost the complete manufacturing site. Such scenarios require additional policies and procedures to ensure secure and safe workspaces compared to conventional automated robot systems which are usually separated from human workers by e.g. safety fences [27, 28].

Besides the serious safety issues in HRC the influence of the robot's behavior on the human collaboration partners also has to be considered. The vulnerable human partner has to trust the system, a feeling that is linked to various factors. Hancock et al.'s study on trust in Human-Robot Interaction (HRI) [29] summarized these factors in three main groups: human-related, robot-related and environmental, and showed that the characteristics and performance of a robot system have the most significant impact on how humans trust such a system (cf. [29]). It is important to allow humans to naturally anticipate the current state and intention of a robot system in order to avoid unexpected behavior triggered by unexpected, dangerous, or novel situations. Dragan *et al.* described a way to incorporate human factors while planning the motion of a robot which could be one way to encode the robot's intention and allow for a more natural partnership that improves the trust and factors [30].

In the end, the final use case of the collaborative application does not really matter. If there is a need to improve the safety, the feeling of trust, or provide a more natural interplay between machines and involved humans, it is crucial to know the state of all involved partners ideally at any operating time. Only if such information can be extracted, concepts like a dynamic and adaptive motion planning [31], or more strict collaborative cell designs e.g. based on safe zones [32] become possible.

1.3 AI in robotics, computer vision and perception

Artificial Intelligence (AI) and machine-learning applications, and in consequence also artificial neural, deep- and convolutional neural networks, have boosted various applications and methods in almost every known research field in the last two decades. This is especially the case when it comes to robotics [33], computer vision [25], and pattern recognition tasks based on almost any digital data source [34, 35, 36, 37]. To perceive the current state of near, middle, and also far surroundings, has, and will probably always be, one of the main research fields within the robotics area (cf. Section 1.2). Any time a robot system has to plan, navigate, interact, or operate in a dynamic area, the best possible knowledge about the targeted environment is required to allow for the safe and secure service of the robot. Therefore in addition to the main focus of this thesis, this Section will give an overview of the different tasks and concrete publications where AI has already pushed robotics applications and certainly will continue to do so in the future.

With the rise and success of many AI technologies, many robotic tasks can finally be tackled with promising and real-world applicable results. Advanced machine learning can be seen as one of the enabling technologies to face various sub-challenges:

- **Robot Dynamics and Control**
  Controlling a robot based on machine learning and a Neural Network (NN) has been a focus in the robotics research area for many years and a lot of interesting approaches to control the bare kinematics have been published decades ago [38, 39, 40, 41]. With modern Deep Neural Network (DNN) techniques, such applications can be heavily expanded as shown by Levine *et al.* [42], by proposing a data-greedy approach to directly learn the abstract relations between a monocular camera system and the industrial robot-arms configuration state, the so called hand-eye coordination. Although very precise robot movement and interaction can now be tackled as demonstrated in many robot assembly applications [43, 44, 45, 46] where machine learning techniques also play a leading role by solving subtasks just as movement, target detection, feedback and also the higher-level navigation tasks [47, 48, 49, 50].

- **Recognition of Objects and Humans**
  The capability to infer positional or other high-level information from e.g. objects and humans (cf. Section 3.3.3) is also an popular and versatile problem in robotics, and is one of the prime examples where machine learning and DNN based methods outperform every system known before [51, 52, 53][3]. Many use cases of CNNs rely on the implementation of 2D regression tasks. So it is not a surprise that many applications from all conceivable domains, e.g. medicine[56], agriculture[57], autonomous driving[58], and many more, used for dense semantic segmentation have also been revolutionized by machine learning [59, 60]. The possibility to extract deep features [61, 62, 63] also had a positive impact on feature-based recognition methods [64, 65, 66].

[3]At the time of the submission of this thesis, all algorithms that lead common object recognition benchmarks [54, 55] employ DNN based methods.

- **Sensor Interpretation and Odometry**
  A vision sensor only produces Two-Dimensional (2D) images, its application is not restricted to that domain. Photogrammetry based methods like structure-from-motion [67] estimates the missing third-dimension of the gathered data and also machine learning-based methods started to revolutionize traditional methods in this field [68, 69]. A sensor's estimated odometry, the movement of the sensor itself, is an important piece of information for such applications. Systems like the *FlowNet* family [70, 71] extract a dense optical flow map of the raw sensor data, which can be used to finally estimate its odometry and improve its accuracy. A different approach to enrich the available sensor data, which would be more or less impossible without employing a CNN, is a direct estimation of a dense depth map as proposed in various publications [72, 73, 74, 75, 76, 77].
- **Calibration and Sensor Fusion**
  When it comes to a basic sensor calibration process itself, AI also offers interesting new possibilities. Calibration is usually a necessary task for every employed sensory setup in order to align the variant information sources in a higher-level system. In order to perform such procedures especially where standard approaches fail, e.g. because of incompatible or missing calibration targets, solutions as *RegNet* [78] or *CalibNet* [79] employ artificial NNs to directly match 3D LiDAR with 2D image data to estimate the relative transformations between each senor. This geometrical relation is crucial in order to fuse various sensor data from different sources.

The mentioned machine learning-based approaches show that modern technologies offer many new prospects to enhance known applications and methods, overcome some of their shortcomings, fuse their information to extend their possibilities, or even make some older approaches obsolete. Nevertheless, this section 1.3 mainly targets to highlight the great improvements machine learning algorithms unlocks and their impact on the field of robotics. With the current rise of all kind of AI-applications, this should be seen as a chance to tackle many unresolved problems, but should not bee seen as a universal remedy. In addition to the machine learning techniques available today, versatile interface technologies that may enable high frequent real-time communication scenarios for a robotic system also make it unnecessary to physically include all thinkable sensory systems on the robot device itself. Many applications, like the here presented approach named *RoPose*, concentrate on extrinsic sensory systems to enrich and fuse the available data with continuous information streams, improving overall perception. The contributions of this thesis, mainly the additional continuous information flow for collaborative robot applications, can thus be used to add important data to the work-flow of many here mentioned solutions and may also help to improve current approaches.

1.4 Contributions of this Thesis

There still are many unresolved questions and challenges regarding the deployment of collaborative robotic environments in manufacturing solutions. This thesis aims to contribute to the big picture by presenting possible solutions and their implementation to the following topics:

- **Data-driven industrial robot pose estimation (*RoPose*).**
  To allow for pose estimation of industrial robot arms within raw 2D RGB-images, the concept of the *RoPose* system [I, II] will be presented. The CNN-based tool helps to extract relevant pose information of visually observed industrial robot arms.

- **Dataset generation based on simulations.**
  For data-driven machine learning applications like *RoPose*, labeled datasets to train and evaluate the system are essential. For development and testing of the main-application, it is also suitable to start with fully synthetic datasets. This thesis shows how to generate such labeled datasets based on Robot Operating System (ROS) [80] and its integrated robotics simulation suite *GAZEBO* [81].

- **Targeted real-world dataset acquisition.**
  Despite the existence of deep- and machine learning approaches fully based on synthetic datasets, there is still a need for real-world datasets, e.g. for evaluation. This thesis shows how to transfer the synthetic dataset generator to real robotic systems to gather important real-world data based on the ROS abstraction concept.

- **Extrinsic camera calibration based on estimated industrial robot poses.**
  An estimated 2D robot pose can also be used as a rich source to extract extrinsic position information of the supervising sensor. The here presented research work shows how to fuse the 2D poses with the 3D positioning information known by the robot controller to derive the extrinsic pose of the sensory system relative to a observed robot.

- **Simultaneous estimation of robot and human poses.**
  In order to expand the *RoPose*-system's scope to collaborative applications this thesis shows how to integrate a camera-based human pose estimation system and so allows for a simultaneous pose-estimation of multiple robots and humans.

2 Background and Basics

“Success is neither magical nor mysterious. Success is the natural consequence of consistently applying the basic fundamentals.”

E. James Rohn

This background chapter is targeted to convey some math and application basics used without further explanation in the remaining thesis. It should be seen as an overview of concepts, approaches, techniques, tools, and physical relations needed to implement the presented applications itself accompanied by some kind of literature review on the respective research field.

2.1 Transformations in Cartesian Coordinate Systems

Transformations of finite, homogeneous coordinates performed in *Cartesian* coordinate systems are very common mathematical tools used in robotics and CV to describe physical relations e.g between rigid objects. As most of the challenges described in this thesis are located in the domains of two- and three-dimensional ($\mathbb{R}^2$ and $\mathbb{R}^3$ resp.) reference systems, this introduction will only discuss these spaces. A rigid body transformation combines translation and rotation information to map different vector spaces and so allows for describing relations of coordinates and poses in different systems. These transformations are usually composed within the so called 2D and 3D *Special Euclidean Groups* ($SE2, SE3$) and the related *Special Orthogonal Groups* ($S02, S03$) further discussed in Section 2.3. The following sections will summarize these concepts and give an overview of the different common representations used in robotics. For the sake of simplicity, the presented examples are mainly visualized in $\mathbb{R}^2$.

2.1.1 Transformations in $\mathbb{R}^2$ and $\mathbb{R}^3$

Transformations can always be described by homogeneous square transformation matrices with the following structure:

$$\boldsymbol{T} = \left(\begin{array}{c|c} \boldsymbol{R}_{n\times n} & \mathbf{t_n} \\ \hline 0 & 1 \end{array}\right),\ \boldsymbol{T} \in SE(n) \tag{2.1}$$

While $\boldsymbol{R}$ is described by an $n \times n$ matrix and contains all information necessary for the rotation, the n-sized vector $\mathbf{t}$ specifies the translation. Eq. 2.2 and 2.3 show the specific transformation matrices for the considered $\mathbb{R}^2$ and $\mathbb{R}^3$.

$$\boldsymbol{T} = \left(\begin{array}{cc|c} r_{00} & r_{01} & x \\ r_{10} & r_{11} & y \\ \hline 0 & 0 & 1 \end{array}\right), \ \boldsymbol{T} \in SE(2) \qquad (2.2)$$

$$\boldsymbol{T} = \left(\begin{array}{ccc|c} r_{00} & r_{01} & r_{02} & x \\ r_{10} & r_{11} & r_{12} & y \\ r_{20} & r_{21} & r_{22} & z \\ \hline 0 & 0 & 0 & 1 \end{array}\right), \ \boldsymbol{T} \in SE(3) \qquad (2.3)$$

To map a specific n-dimensional coordinate vector $\mathbf{p}$ to another coordinate system with known relations, it is possible to multiply the $n \times n$ transformation matrix with the homogenized point of interest from the left (Eq. 2.4), or by employing the transformation information separately like shown in Eq. 2.5.

$$\begin{pmatrix} \mathbf{p}' \\ 1 \end{pmatrix} = \boldsymbol{T} \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix} \qquad (2.4)$$

$$\mathbf{p}' = \boldsymbol{R}\mathbf{p} + \mathbf{t} \qquad (2.5)$$

The inversion of a square transformation matrix $\boldsymbol{T}^{-1}$ describes also the inverse transformation operation. A defined transformation can so be used to transform vectors in both directions.

$$\begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix} = \boldsymbol{T}^{-1} \begin{pmatrix} \mathbf{p}' \\ 1 \end{pmatrix} \qquad (2.6)$$

2.1.2 Translation

An n-dimensional translation vector $\mathbf{t}$ contains the positional information of a coordinate or point for each of the available coordinate base axes.

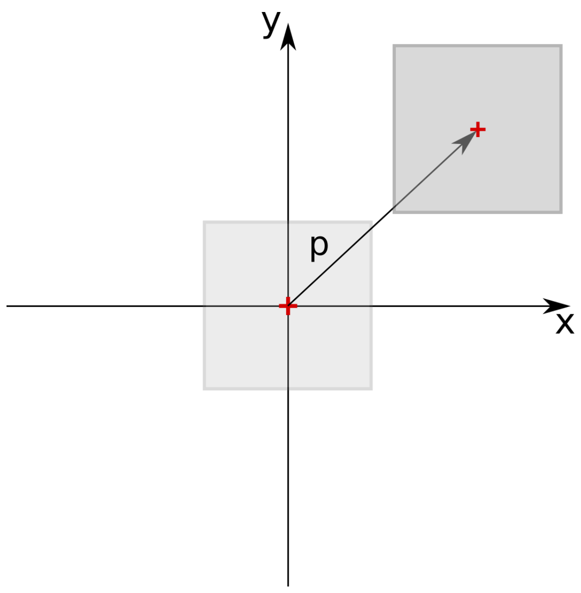


Figure 2.1: Visualization of a translation transformation in $\mathbb{R}^2$.

$$\mathbf{p} = \begin{pmatrix} x \\ y \end{pmatrix}, \ \mathbf{p} \in \mathbb{R}^2 \qquad (2.7) \qquad\qquad \mathbf{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \ \mathbf{p} \in \mathbb{R}^3 \qquad (2.8)$$

2.1.3 Rotation and Orientation Representations

Rotation operations (cf. Figure 2.2) are more complex. Especially when it comes to higher-dimensional coordinate spaces like $\mathbb{R}^3$ several commonly used representations and their connected notations, concepts and philosophies have to be considered. The most common ones are explained in the following sections.

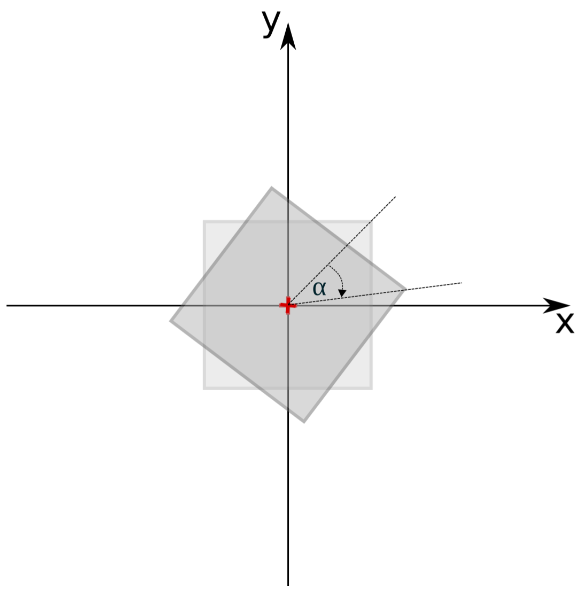


Figure 2.2: Visualization of a rotation transformation, based on a single angle α, in $\mathbb{R}^2$.

2.1.3.1 Rotation Matrices

A rotation matrix contains the needed information about the rotation in its rawest form. As described in 2.1.1, this matrix is also part of the transformation matrix itself and every here presented representation can be converted back into this raw matrix form. Related to the Degrees of Freedom (DoF) of the respective vector spaces, the particular rotation matrices are defined as follows:

$$\boldsymbol{R} = \begin{pmatrix} r_{00} & r_{01} \\ r_{10} & r_{11} \end{pmatrix}, \ \boldsymbol{R} \in SO(2) \qquad (2.9) \qquad \boldsymbol{R} = \begin{pmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{pmatrix}, \ \boldsymbol{R} \in SO(3) \qquad (2.10)$$

2.1.3.2 Rotations in $\mathbb{R}^2$

In 2D coordinate systems, an orientation information has just one DoF and thus is a single angle parameter α. The associated rotation matrix can be calculated as follows:

$$\boldsymbol{R}(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}, \boldsymbol{R}(\alpha) \in SO(2) \tag{2.11}$$

2.1.3.3 Chained rotations

To define an orientation in $\mathbb{R}^3$ it is possible to chain multiple individual rotations around the single axis of a coordinate system. For each axis of an orthogonal coordinate system a specific rotation matrix can be defined (cf. Equations 2.12-2.14). This formulation offers the possibility to store and share all rotation related information in $\mathbb{R}^3$ with just three angles [82].

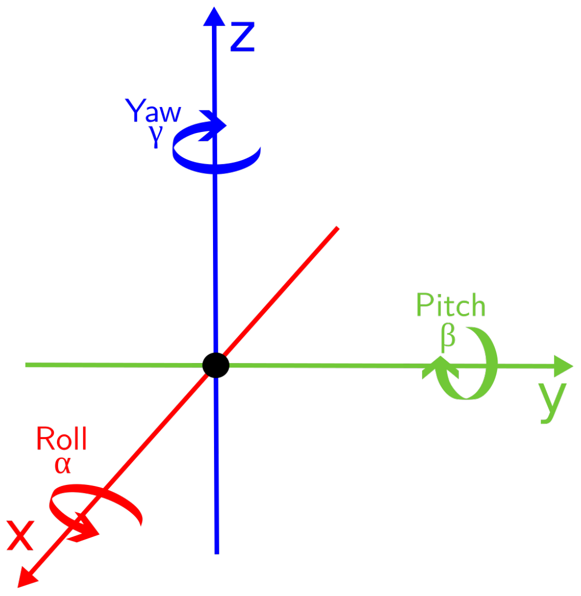


Figure 2.3: Visualization of the possible rotations around each base vector of a $\mathbb{R}^3$ coordinate system and the corresponding Euler angles α, β, and γ, respectively yaw, pitch, and roll convention.

$$\boldsymbol{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}, \ \boldsymbol{R}_x(\alpha) \in SO(3) \tag{2.12}$$

$$\boldsymbol{R}_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}, \ \boldsymbol{R}_y(\beta) \in SO(3) \tag{2.13}$$

$$\boldsymbol{R}_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}, \ \boldsymbol{R}_z(\gamma) \in SO(3) \tag{2.14}$$

One of the most common conventions to perform chained rotations are the Euler angles, first introduced by Leonard Euler in 1776 [83]. Euler angles consist of three unique rotations around the X-, Y-, and Z-axis (α, β, and γ). This rotation operation is designated as intrinsic because every single rotation rotates the coordinate axis itself. This notation allows twelve unique sequences to conduct the single rotations (cf. [84, 82]) and to compose the final rotation matrix which highly depends on the chosen order. Based on the common *XYZ*-order, the rotation matrix can be composed as follows:

$$\boldsymbol{R}_{xyz} = \boldsymbol{R}_x(\alpha)\boldsymbol{R}_y(\beta)\boldsymbol{R}_z(\gamma), \text{ with } \boldsymbol{R}_{xyz} \in SO(3) \tag{2.15}$$

$$\boldsymbol{R}_{xyz} = \begin{pmatrix} \cos\beta\cos\gamma & -\cos\beta\sin\gamma & \sin\beta \\ \cos\alpha\sin\gamma + \sin\alpha\sin\beta\cos\gamma & \cos\alpha\sin\gamma - \sin\alpha\sin\beta\sin\gamma & -\sin\alpha\cos\beta \\ \cos\alpha\sin\gamma - \cos\alpha\sin\beta\cos\gamma & \sin\alpha\cos\gamma + \cos\alpha\sin\beta\sin\gamma & \cos\alpha\cos\beta \end{pmatrix} \tag{2.16}$$

An almost identical representation is the well known roll, pitch, and yaw (*RPY*) notation. The *RPY* angles also decode the rotation in three sub rotations around one absolute angle. In contrast to Euler angles, the *RPY* convention is designated as extrinsic and each rotation is defined with respect to a static reference frame. However, the corresponding matrix operations and the final resulting rotation matrices are calculated the same way and following the Euler angle convention (cf. Eq. 2.15 and 2.16).

Minimal representations with a list of parameters of just three single values can lead to singularities within the covered domain. A so called *gimbal lock* [85] occurs when two of the three coordinate axes remain in a parallel relation after applying the first two rotations (cf. Eq. 2.15). This leads to the loss of one DoF and will result in wrong rotation matrices. The *gimbal lock* occurs frequently in many applications and should always be considered when working with pure Euler angles.

2.1.3.4 Axis-Angles

The axis-angle representation decodes the $\mathbb{R}^3$ rotation information based on a three dimensional axis of rotation unit vector $\mathbf{n} = [n_x, n_y, n_z]^T$ $(||\mathbf{n}|| = 1)$ and a dedicated angle θ as shown in Figure 2.4

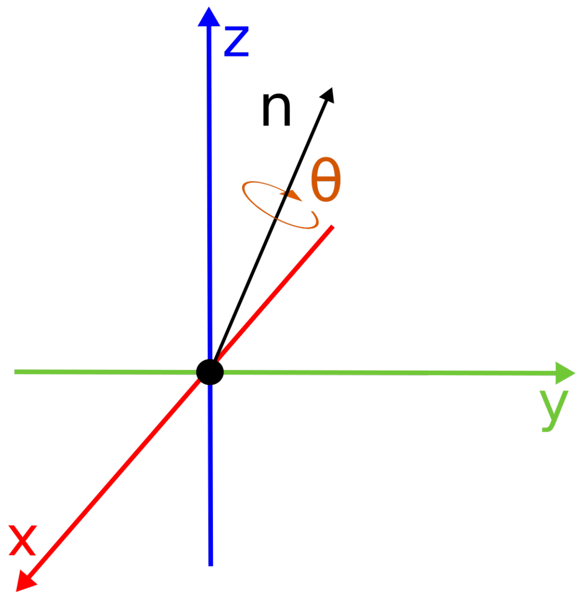


Figure 2.4: Visualization of the axis-angles notation in $\mathbb{R}^3$. The rotation is described with the three-dimensional unit vector $\mathbf{n}$ (axis of rotation) and the corresponding angle θ.

To apply an axis-angle rotation to a known vector of interest $\mathbf{x}$, the *Euler–Rodrigues formula* [86] is utilized as follows:

$$\mathbf{x}' = \mathbf{x} + (\sin\theta)\mathbf{n} \times \mathbf{x} + (1 - \cos\theta)\mathbf{n} \times (\mathbf{n} \times \mathbf{x}) \qquad (2.17)$$

2.1.3.5 Quaternions

Quaternions are usually the most common, but also the most abstract representation in computer graphics and robotics because they offer a compact, efficient, and stable way to deal with rotations. Quaternions are generally composed with the help of four parameters. One element (a) representing the real part or scalar part of the quaternion, and three complex components representing the imaginary part or vector part (b, c, d):

$$q = a + bi + cj + dk \tag{2.18}$$

Each imaginary component encodes together with the shared real part a rotation along the axes of the Euclidean $\mathbb{R}^3$ space.

To properly represent rotations, only normalized quaternions, the so called *unit quaternions*, should be used.

$$q' = \frac{q}{||q||} \tag{2.19}$$
$$||q|| = \sqrt{a^2 + b^2 + c^2 + d^2} \tag{2.20}$$

The easiest, but not only way to encode known rotation information in quaternions is to use the physical rotation representation of the axis angles (cf. Section 2.1.3.4) and use the following relations (cf. [82] Eq. 175):

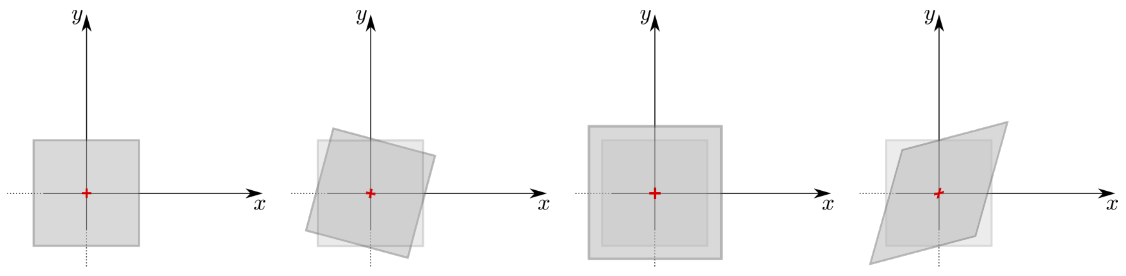
$$a = q_a = \cos\frac{\theta}{2} \tag{2.21}$$
$$b = q_b = n_0 \sin\frac{\theta}{2} \tag{2.22}$$
$$c = q_c = n_1 \sin\frac{\theta}{2} \tag{2.23}$$
$$d = q_d = n_2 \sin\frac{\theta}{2} \tag{2.24}$$

It is of note that quaternions are mainly a mathematical construct and thus do not necessarily need to have an explicit physical meaning, but can be applied to rotations. A formal introduction to quaternions and their particular usage for transformations respectively rotations was published by Kuipers *et al.* [84].

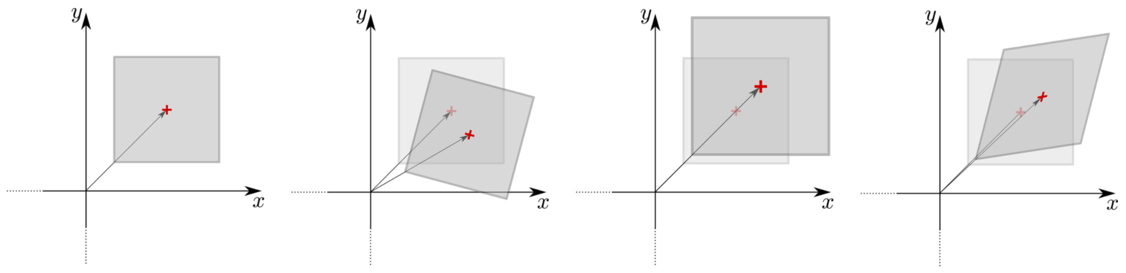
2.1.4 Affine Transformations

Besides the already presented basic transformations, i.e. translations and rotations (cf. Sections 2.1.2 and 2.1.3), advanced transformations to manipulate whole images, objects or single coordinates exists. Among other tasks, the so called affine transformations are commonly used to augment 2D data, such as images, e.g. by transforming every single pixel and also the corresponding ground truth data for image-based machine learning applications.

As these transformations are usually used as extrinsic operations and will be performed based on the origin of the global reference frame, the position of the transformation target within that frame should always be considered. Figure 2.5 shows the influence of the particular reference system to rotation, scale, and shear transformations.



(a) Affine transformations within target frame.



(b) Affine transformations within global reference frame.

Figure 2.5: Effect of a global reference frame to affine transformations. From left: Origin state, rotation, scale, shear.

Due to several image representations, it might be necessary to translate the image center to the origin of a defined global coordinate frame (c.f. 2.5a) before performing the manipulation and restore the original translation afterward. Please note that this problem is not restricted to affine image manipulation and can also occur when working with rigid body transformations in $\mathbb{R}^3$. These affine transformations can formally be applied with homogeneous transformation matrices as described in Section 2.1.1.

2.1.4.1 Scale

A scaling operation is used to enlarge or shrink a given target by a given factor in the particular axis. Figure 2.6 visualizes the scaling and Equations 2.26.

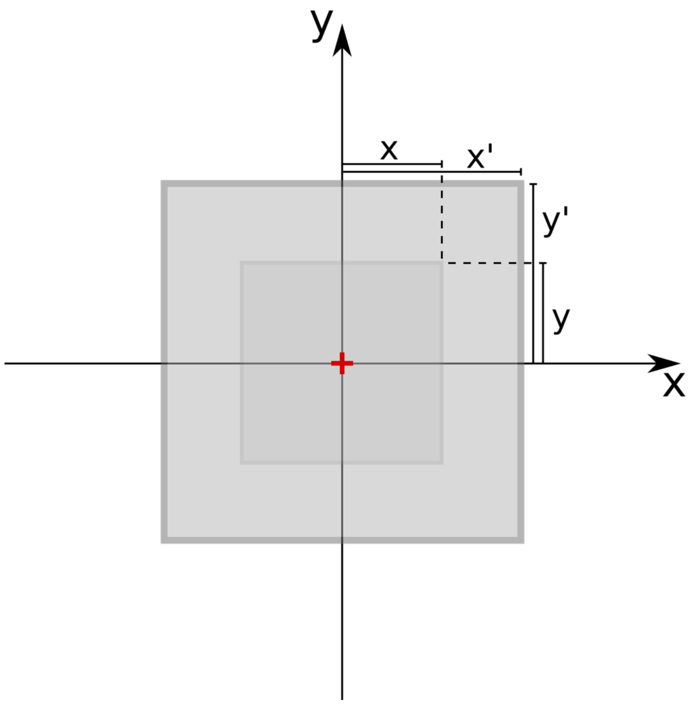


Figure 2.6: Visualization of a scale transformation in $\mathbb{R}^2$.

To finally perform a scaling, the scaling factors (s_{xyz}) have to be calculated as shown in the following example for the x-axis and then can be used within the transformation matrix as shown in Eq. 2.26 resp. 2.27:

$$s_x = \frac{x'}{x} \tag{2.25}$$

$$\boldsymbol{T}_s = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}, \text{ with } s_{x,y} \geq 0 \tag{2.26}$$

$$\boldsymbol{T}_s = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}, \text{ with } s_{x,y,z} \geq 0 \tag{2.27}$$

2.1.4.2 Shear

The shearing operation translates each target in a fixed direction by a distance proportional to their perpendicular distance (sh). Figure 2.7 visualizes the scaling and the Equations 2.28 - 2.29 specify the matrices to apply the shearing.

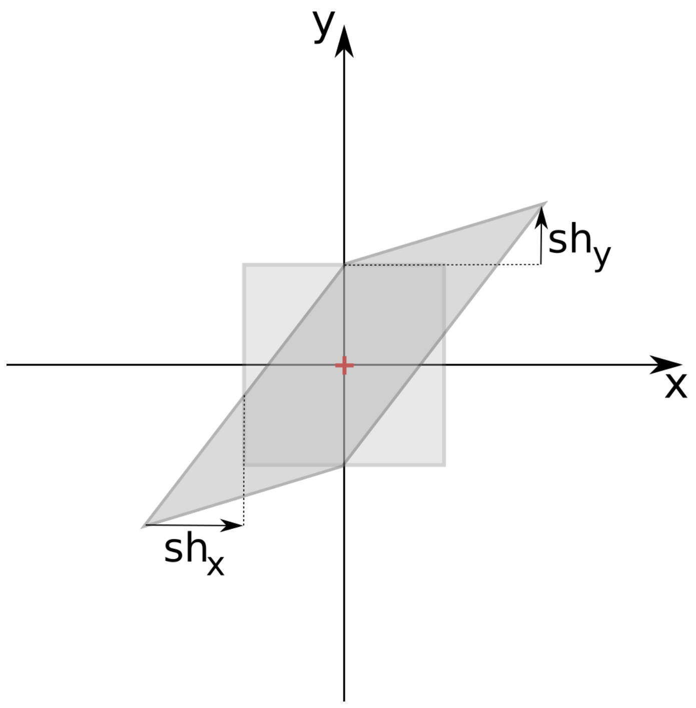


Figure 2.7: Visualization of a shear transformation in $\mathbb{R}^2$. With $sh_{x,y} > 0$

$$\boldsymbol{T}_{sh} = \begin{pmatrix} 1 & sh_y \\ sh_x & 1 \end{pmatrix} \tag{2.28}$$

$$\boldsymbol{T}_{sh} = \begin{pmatrix} 1 & sh_{xy} & sh_{xz} \\ sh_{yx} & 1 & sh_{yz} \\ sh_{zx} & sh_{zy} & 1 \end{pmatrix} \tag{2.29}$$

Each shearing parameter (sh) in Eq. 2.28 describes the shearing relative to the particular axis or the relative plane in $\mathbb{R}^2$ and Eq. 2.29 in $\mathbb{R}^3$.

2.1.4.3 Combined Affine Transformations

Sometimes it is necessary to combine multiple sub transformations to one affine transformation ($\boldsymbol{T}_c$) and apply the resulting manipulation with just one operation. To combine all single transformations the sequential dot-product of the single transformations is utilized.

$$\boldsymbol{T}_c = \boldsymbol{T}_{rot} \cdot \boldsymbol{T}_{shear} \cdot \boldsymbol{T}_{scale} \cdot \boldsymbol{T}_{trans} \tag{2.30}$$

2.2 Spherical Coordinate Systems

Within the versatile field of robotics, it might be useful to organize basic positional information in other parameterized coordinate systems than the well known basic n-dimensional Cartesian systems. One example of such a system is a spherical coordinate system as shown in Figure 2.8.

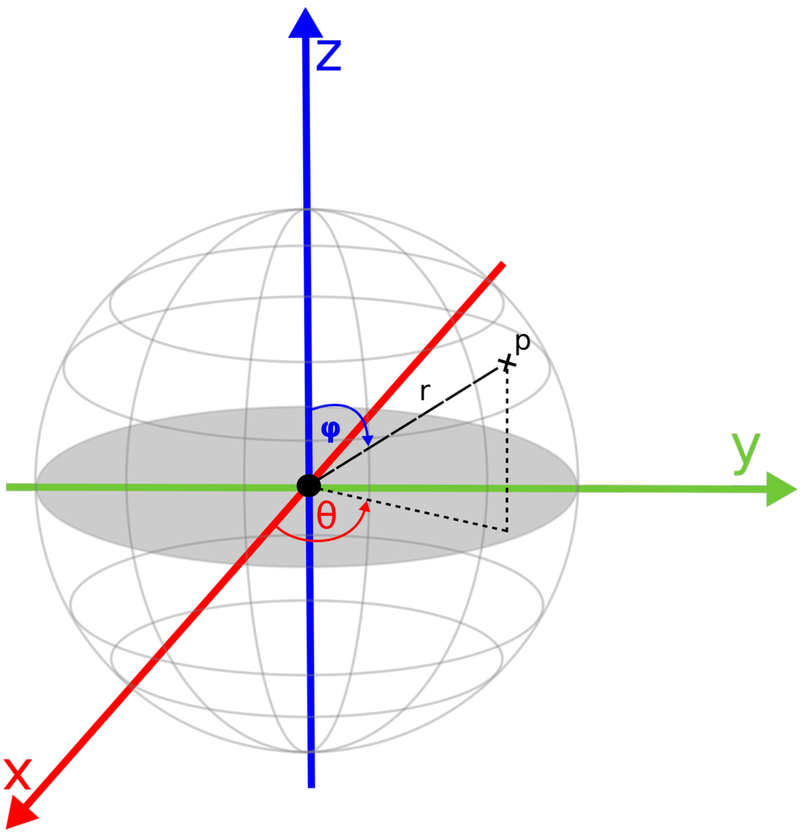


Figure 2.8: Visualization of spherical coordinates in $\mathbb{R}^3$. r: radius, φ: polar angle (*inclination*), θ: azimuthal angle (*azimuth*).

Spherical systems can be used to describe a three dimensional vector $\mathbf{p}$ based on a radius r, a polar angle (*inclination*) φ and the azimuthal angle (*azimuth*) θ. The following Equations can be used to translate coordinates from a standard Cartesian based system to a spherical, and vice versa.

$$r = \sqrt{x^2 + y^2 + z^2} \tag{2.31}$$
$$\theta = \arctan \frac{y}{x} \tag{2.32}$$
$$\varphi = \arccos \frac{z}{r} \tag{2.33}$$
$$x = r \, \sin\theta \, \cos\varphi \tag{2.34}$$
$$y = r \, \sin\theta \, \sin\varphi \tag{2.35}$$
$$z = r \, \cos\varphi \tag{2.36}$$

These equations may differ if the relative axes for the inclination and azimuthal angles changes. Within this work, the polar angle φ describes the angle between the Z-axis while θ refers to the angle between the X-axis and the vector $\mathbf{p}$.

2.3 Lie Groups

In the field of Computer Vision (CV) and especially robotics, the transformations introduced in Section 2.1.1 are often described based on the concept of *Lie Groups*. Each Lie group is a topological group containing specific elements and can also be seen as a differentiable manifold. This representation offers a more abstract view on transformations and becomes very practical when there is a need to look at the derivatives of transformation and also their chronological sequence.
The relevant groups to describe transformations in $\mathbb{R}^2$ and $\mathbb{R}^3$, and therefore the only groups this document focuses on, are:

- **Special Orthogonal Group in $\mathbb{R}^2$ - $SO(2)$**
  The $SO(2)$ group contains all rotation matrices $R_{2\times2}$ in $\mathbb{R}^2$ (cf. Section 2.1.3).

- **Special Orthogonal Group in $\mathbb{R}^3$ - $SO(3)$**
  The $SO(3)$ group contains all rotation matrices $R_{3\times3}$ in $\mathbb{R}^3$ (cf. Section 2.1.3).

- **Special Euclidean Group in $\mathbb{R}^2$ - $SE(2)$**
  The $SE(2)$ group contains all homogeneous transformation matrices $T_{3\times3}$ in $\mathbb{R}^2$ (cf. Section 2.1.1).

- **Special Euclidean Group in $\mathbb{R}^3$ - $SE(3)$**
  The $SE(3)$ group contains all homogeneous transformation matrices $T_{4\times4}$ in $\mathbb{R}^3$ (cf. Section 2.1.1).

In addition to basic mathematical introductions to Lie groups and their usage preciously reported in [87, 88], both Ethan Eade [89] and Jose-Luis Blanco [90] provide remarkable technical reports focusing on concrete implementations of groups. The here presented groups are another way to deal with transformations and can be seen as a substitution to more traditional methods that comes with useful mathematical properties for many applications and also simplifies the implementation. Especially when looking at the parameter space for each group, it becomes apparent, that the final physical connections and relations, basically all rigid body transformations, are identical.

2.3.1 Basic Definitions

Before discussing the particular groups in detail some basics, commonalities, and properties need to be clarified. Each Lie group (G) has an associated *Lie algebra* ($\mathfrak{g}$) that is defined to be the tangent space at the Lie group's unit element (I). A Lie algebra is defined by linear combinations of the so called *generators* (see examples in Sections 2.3.2-2.3.5) and the particular parameters (P), corresponding to the dimension of the group. Within the here discussed groups, the Lie algebra are always a set of skew symmetric matrices ($\boldsymbol{\omega}_\times$). Because the $SO(2)$ group (cf. Section 2.3.2) probably is the most tangible group, the following Figure 2.9 visualizes the mentioned elements based on the unit circle.

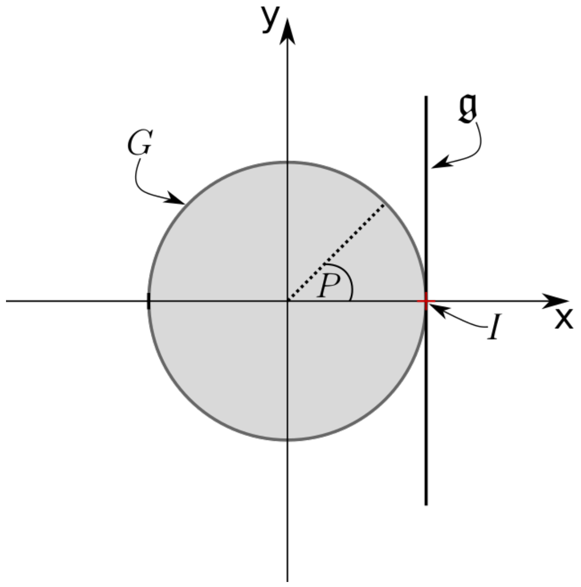


Figure 2.9: Visualization of the $SO(2)$ group. Besides the group G itself, which is defined by the unit circle, the associated Lie algebra $\mathfrak{g}$ is the tangent at the unit element I. Because the parameter space of $SO(2)$ is one dimensional, the parameters (P) consists of just one element describing a single rotation angle.

Several operators and actions are available when working with Lie groups:

- **The *Hat*-operator $\hat{\cdot}$**
  The *Hat*-operator is a vector-space isomorphism from an n-dimensional real vector space to the Lie Algebra. In our case it maps specific parameters (P) of a group element to skew symmetric matrices $\boldsymbol{\omega}_\times$.

- **The *Vee*-operator $\vee$**
  The *Vee*-operator is the inverse of the *Hat*-Operator and maps a Lie algebra to specific parameter sets of a Lie group's parameter space.

  In every group explained in this document, the skew symmetric matrix form of each Lie algebra is strictly defined. The *Vee* operation maps given parameters to the corresponding elements of the matrix. For the sake of completeness, one example is given in the explanation of the SO(3) group in Section 2.3.3.3.

  Chirikjian *et al.* gives in Eq. 10.31 [87] the following general formula for the *Vee*-operator to map the given parameters $(x_1 \ldots\ x_n)$ within an n-dimensional group.

$$\left(\sum_{i=1}^{n} x_i E_i\right)^{\vee} \doteq (x_1\ x_2\ x_3\ \ldots\ x_n)^T \tag{2.37}$$

  Note that the E_i components corresponds to the generators (cf. definitions of the special groups in Section 2.3.2 - 2.3.5) of the respective group.

- **The *exponential map***
  The *exponential map* is used to map an element of the Lie algebra to group elements.

$$exp : \mathfrak{g} \mapsto G \tag{2.38}$$

  It is formally defined with the matrix exponential of a skew symmetric matrix $(\boldsymbol{\omega}_\times)$ defining a Lie algebra which can be calculated employing the following power series (cf. [91]):

$$e^{\boldsymbol{\omega}_\times} = \sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{\omega}_\times^k \tag{2.39}$$

- **The *logarithmic map***
  The *logarithmic map* defines the inverse of the *exponential map* and maps group elements to the Lie algebra $(\mathfrak{g})$.

$$log : G \mapsto \mathfrak{g} \tag{2.40}$$

2.3.2 Special Orthogonal Group in $\mathbb{R}^2$ - $SO(2)$

The $SO(2)$ group contains all 2D rotation matrices ($\boldsymbol{R}$) as described in Section 2.1.3.2.

2.3.2.1 Generators and Parameters of $SO(2)$

All rotations in $\mathbb{R}^2$ can be described by a single angle ($\alpha, with 0 \leq \alpha < 2\pi$). Thus the resulting group parameter list (P_{SO2}) is also one dimensional.

$$P_{SO2} = \alpha \tag{2.41}$$

The one dimensional Lie algebra is given by a linear combination of α and the single *generator* $\boldsymbol{G}_0$. Formally, the sole generator for the Lie algebra $so(2)$ encodes the differential rotation information and can be obtained by differentiating the standard rotation matrix (cf. Eq. 2.11) and evaluating it at $\alpha = 0$ which is also called the identity element:

$$\boldsymbol{G}_0 = \left.\frac{\partial \boldsymbol{R}(\alpha)}{\partial \alpha}\right|_{\alpha=0} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \tag{2.42}$$

2.3.2.2 Hat-Operator of $SO(2)$

As the hat operation only maps the parameters to the generators, this can be performed with a multiplication:

$$\hat{P_{SO2}} = \alpha \boldsymbol{G}_0 = \begin{pmatrix} 0 & -\alpha \\ \alpha & 0 \end{pmatrix} = \boldsymbol{\omega}_\times,\ \boldsymbol{\omega}_\times \in so(2) \tag{2.43}$$

2.3.2.3 Exponential map of $SO(2)$

The final power series of the matrix exponential of $\boldsymbol{\omega}_\times$ can be reduced to the following equation (cf. [89] Eq. 106) and results in a rotation matrix $\boldsymbol{R}$:

$$exp(\boldsymbol{\omega}_\times) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} = \boldsymbol{R},\ \boldsymbol{R} \in SO(2) \tag{2.44}$$

2.3.2.4 Logarithmic map of $SO(2)$

The logarithmic map to transform a 2D rotation matrix $\boldsymbol{R}$ to a Lie algebra can be calculated by first extracting the explicit angle α (cf. Eq. 2.41) and applying an additional Hat operation (cf. [89] Eq. 108):

$$\alpha = \arctan(\boldsymbol{R}_{10}, \boldsymbol{R}_{00}) \tag{2.45}$$

$$\ln(\boldsymbol{R}) = \hat{\alpha},\ \hat{\alpha} \in so(2) \tag{2.46}$$

2.3.3 Special Orthogonal Group in $\mathbb{R}^3$ - $SO(3)$

The $SO(3)$ group contains all possible rotation matrices ($\boldsymbol{R}$) in $\mathbb{R}^3$ as detailed in Section 2.1.3.

2.3.3.1 Generators and Parameters of $SO(3)$

As all of these rotations can be described by a chained rotation along each axis the resulting parameter space $\mathbf{P}_{SO3}$ for the $SO(3)$ group has three dimensions, containing one rotation angle for each of the available coordinate axis (x-axis $=\alpha$, y-axis $=\beta$, z-axis $=\gamma$, with $0 \leq \alpha, \beta, \gamma < 2\pi$). Geometrically this parametrization is related to the already mentioned Axis Angles (cf. Section 2.1.3.4) but with $\theta = \sqrt{\alpha^2 + \beta^2 + \gamma^2}$.

$$\mathbf{P}_{SO3} = (\alpha, \beta, \gamma)^T \tag{2.47}$$

To generate the Lie algebra $so(3)$ for the $SO(3)$ group the following generator matrices are used. Geometrically these matrices correspond to the differentiated $\mathbb{R}^3$ rotation matrices (cf. Eq. 2.12 - 2.14) all evaluated at the identity ($\alpha = \beta = \gamma = 0$).

$$\boldsymbol{G}_0 = \left.\frac{\partial \boldsymbol{R}_x(\alpha)}{\partial \alpha}\right|_{\alpha=0} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \tag{2.48}$$

$$\boldsymbol{G}_1 = \left.\frac{\partial \boldsymbol{R}_y(\beta)}{\partial \beta}\right|_{\beta=0} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix} \tag{2.49}$$

$$\boldsymbol{G}_2 = \left.\frac{\partial \boldsymbol{R}_z(\gamma)}{\partial \gamma}\right|_{\gamma=0} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.50}$$

2.3.3.2 Hat-Operator of $SO(3)$

The hat operator maps a vector of parameters (α, β, γ) to a skew matrix. It is defined by the linear combination of each parameter and the generators:

$$\hat{\mathbf{P}}_{SO3} = \alpha \boldsymbol{G}_0 + \beta \boldsymbol{G}_1 + \gamma \boldsymbol{G}_2 = \begin{pmatrix} 0 & -\gamma & \beta \\ \gamma & 0 & -\alpha \\ -\beta & \alpha & 0 \end{pmatrix} = \boldsymbol{\omega}_\times, \quad \text{with } \boldsymbol{\omega}_\times \in so(3) \tag{2.51}$$

2.3.3.3 Vee-Operator of $SO(3)$

As mentioned before, the Vee operation is equivalent to a direct mapping of some entries of a skew symmetric matrix to the parameters and can be easily performed on the special groups detailed in this work. The parameters can be mapped directly from a Lie algebra $\boldsymbol{\omega}_\times$ by extracting the single matrix elements:

$$\mathbf{P}_{SO3} = (\boldsymbol{\omega}_{\times 21}, \boldsymbol{\omega}_{\times 02}, \boldsymbol{\omega}_{\times 10})^T = (\alpha, \beta, \gamma)^T \tag{2.52}$$

2.3.3.4 Exponential map of $SO(3)$

The final power series of the matrix exponential of $\boldsymbol{\omega}_\times$ is given by the *Rodrigues* formula (cf. Section 2.1.3.4 and [92] Eq. (1.2.6)) which results directly in an element of the group and thus a rotation matrix R.

$$exp(\boldsymbol{\omega}_\times) = \boldsymbol{I}_3 + \left(\frac{\sin\theta}{\theta}\right)\boldsymbol{\omega}_\times + \left(\frac{1-\cos\theta}{\theta^2}\right)\boldsymbol{\omega}_\times^2 = \boldsymbol{R}, \text{ with } \boldsymbol{R} \in SO(3) \tag{2.53}$$

with I_3 = symmetric identity matrix with size 3×3,
and $\theta = \sqrt{\alpha^2 + \beta^2 + \gamma^2} = ||\mathbf{P}_{SO3}||$

2.3.3.5 Logarithmic map of $SO(3)$

The logarithmic map for $SO(3)$ is defined by the inverse of the exponential map (cf. [89] Eq. 17-18):

Note that $\text{tr}(\boldsymbol{R})$ is the trace operator of square matrices and is defined by the sum of all main diagonal elements (cf. [93]).

$$\theta = \arccos\left(\frac{\mathrm{tr}(\boldsymbol{R}) - 1}{2}\right) \tag{2.54}$$

$$\ln(\boldsymbol{R}) = \frac{\theta}{2\sin\theta}(\boldsymbol{R} - \boldsymbol{R}^T) \tag{2.55}$$

2.3.4 Special Euclidean Group in $\mathbb{R}^2$ - $SE(2)$

The SE(2) group unites all possible transformations in $\mathbb{R}^2$. This includes all rotation matrices available in SO(2) and extends this set with all possible 2D translations.

2.3.4.1 Generators and Parameters of $SE(2)$

The list of parameters consists of the two individual translations parameters (t_x, t_y), and a parameter α as the rotation angle.

$$\mathbf{P}_{SE2} = (t_x, t_y, \alpha)^T \tag{2.56}$$

The needed generators also combine the differential translation ($\boldsymbol{G}_0$ and $\boldsymbol{G}_1$) and rotation operations, which is identical to the sole generator of the $SO(2)$ group (see Eq. 2.42). These generators relate to the partial derivatives of the transformation matrix T detailed in Eq. 2.2, again evaluated at the identity.

$$\boldsymbol{G}_0 = \left.\frac{\partial \boldsymbol{T}}{\partial x}\right|_{x=0} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.57}$$

$$\boldsymbol{G}_1 = \left.\frac{\partial \boldsymbol{T}}{\partial y}\right|_{y=0} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.58}$$

$$\boldsymbol{G}_2 = \left.\frac{\partial \boldsymbol{T}}{\partial \alpha}\right|_{\alpha=0} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.59}$$

2.3.4.2 Hat-Operator of $SE(2)$

The parameters can be mapped to the Lie algebra with the help of the generators $\boldsymbol{G}_0$ - $\boldsymbol{G}_2$.

$$\hat{\mathbf{P}}_{SE2} = t_x\boldsymbol{G}_0 + t_y\boldsymbol{G}_1 + \alpha\boldsymbol{G}_2 \tag{2.60}$$

$$\begin{pmatrix} 0 & -\alpha & t_x \\ \alpha & 0 & t_y \\ 0 & 0 & 0 \end{pmatrix} = \left(\begin{array}{c|c} \boldsymbol{\omega}_{\times R} & \mathbf{t} \\ \hline 0 & 0 \end{array}\right) = \boldsymbol{\omega}_\times, \text{ with } \boldsymbol{\omega}_\times \in se(2) \tag{2.61}$$

2.3.4.3 Exponential map of $SE(2)$

To calculate the exponential map in $SE(2)$ this operation is split into two parts. First the logarithmic map of the rotation part $\boldsymbol{\omega}_{\times R}$ is calculated as known from the $SO(2)$ group (cf. Section 2.3.2.3), and then the final combined exponential map can be calculated according to the following equations (cf. [89] Eq. 120-136):

$$\theta = \arctan(\boldsymbol{R}_{10}, \boldsymbol{R}_{00}) \tag{2.62}$$

$$\boldsymbol{V} = \frac{1}{\theta}\begin{pmatrix} \sin\theta & -1+\cos\theta \\ 1-\cos\theta & \sin\theta \end{pmatrix} \tag{2.63}$$

$$exp(\boldsymbol{\omega}_\times) = \left(\begin{array}{c|c} \exp(\boldsymbol{\omega}_{\times R}) & \boldsymbol{V}\mathbf{t} \\ \hline 0 & 1 \end{array}\right) = \left(\begin{array}{c|c} \boldsymbol{R} & \mathbf{t} \\ \hline 0 & 1 \end{array}\right) = \boldsymbol{T}, \text{ with } \boldsymbol{T} \in SE(2) \tag{2.64}$$

where θ can be extracted from $\exp(\boldsymbol{\omega}_{\times R})$ according to Eq. 2.62 (cf. Eq. 2.45).

2.3.4.4 Logarithmic map of $SE(2)$

For the logarithmic mapping it is easier to operate on the parameter space of $SE(2)$. While $\boldsymbol{V}^{-1}$ injects the rotation information into the translation part (cf.[89] Eq. 135), the rotation parameter θ can again be extracted from $\boldsymbol{R}$ according to Eq. 2.45.

$$\boldsymbol{V}^{-1} = \frac{\theta}{\sin^2\theta + (1-\cos\theta)^2}\begin{pmatrix} \sin\theta & 1-\cos\theta \\ -1+\cos\theta & \sin\theta \end{pmatrix} \tag{2.65}$$

$$p'_{SE(2)} = \begin{pmatrix} \boldsymbol{V}^{-1}t \\ \theta \end{pmatrix} \tag{2.66}$$

$$\ln\left(\begin{array}{c|c} \boldsymbol{R} & \mathbf{t} \\ \hline 0 & 1 \end{array}\right) = \hat{p}'_{SE(2)} = \boldsymbol{\omega}_\times, \text{ with } \boldsymbol{\omega}_\times \in se(2) \tag{2.67}$$

2.3.5 Special Euclidean Group in $\mathbb{R}^3$ - $SE(3)$

The SE(3) group contains all possible transformations in $\mathbb{R}^3$. This includes all rotation matrices available in SO(3) and extends the set with all possible translations within a 3D system.

2.3.5.1 Generators and Parameters of $SE(3)$

The resulting parameter space $\mathbf{P}_{SE3}$ for the $SE(3)$ group has six dimensions, containing a translation parameter and one rotation angle for each of the available coordinate axes.

$$\mathbf{P}_{SE3} = (t_x, t_y, t_z, \alpha, \beta, \gamma)^T \tag{2.68}$$

All transformations in $\mathbb{R}^3$, i.e. all members of $SE(3)$ can be described with a symmetric 4×4 matrix (cf. Section 2.1.1), and therefore the employed generators and the resulting Lie algebra have the same dimensions. Again, the respective generators of the Lie algebra $se(3)$ directly corresponds to the partial derivatives of the $\mathbb{R}^3$ transformation matrix (Eq. 2.3) evaluated at the identity.

$$\boldsymbol{G}_0 = \left.\frac{\partial \boldsymbol{T}}{\partial x}\right|_{x=0} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.69}$$

$$\boldsymbol{G}_1 = \left.\frac{\partial \boldsymbol{T}}{\partial y}\right|_{y=0} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.70}$$

$$\boldsymbol{G}_2 = \left.\frac{\partial \boldsymbol{T}}{\partial z}\right|_{z=0} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.71}$$

$$\boldsymbol{G}_3 = \left.\frac{\partial \boldsymbol{T}}{\partial \alpha}\right|_{\alpha=0} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.72}$$

$$\boldsymbol{G}_4 = \left.\frac{\partial \boldsymbol{T}}{\partial \beta}\right|_{\beta=0} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.73}$$

$$\boldsymbol{G}_5 = \left.\frac{\partial \boldsymbol{T}}{\partial \gamma}\right|_{\gamma=0} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.74}$$

2.3.5.2 Hat-Operator of $SE(3)$

The linear combination of the parameters and the generators define the hat operation and lead to the skew symmetric matrix:

$$\hat{\mathbf{P}}_{SE3} = t_x\boldsymbol{G}_0 + t_y\boldsymbol{G}_1 + t_z\boldsymbol{G}_2 + \alpha\boldsymbol{G}_3 + \beta\boldsymbol{G}_4 + \gamma\boldsymbol{G}_5 = \boldsymbol{\omega}_\times \tag{2.75}$$

$$\boldsymbol{\omega}_\times = \begin{pmatrix} 0 & -\gamma & \beta & t_x \\ \gamma & 0 & -\alpha & t_y \\ -\beta & \alpha & 0 & t_z \\ 0 & 0 & 0 & 0 \end{pmatrix} = \left(\begin{array}{c|c} \boldsymbol{\omega}_{\times R} & \mathbf{t} \\ \hline 0 & 0 \end{array}\right), \text{ with } \boldsymbol{\omega}_\times \in se(3) \tag{2.76}$$

2.3.5.3 Exponential map of $SE(3)$

Like in $SE(2)$ the exponential mapping of the rotation part of the skew symmetric matrix can be isolated as shown in Eq. 2.64 based on the matrix $\boldsymbol{V}$ (cf. [90] Eq. 9.22):

$$\boldsymbol{V} = \boldsymbol{I}_3 + \frac{1-\cos\theta}{\theta^2}\omega_{\times R} + \frac{\theta - \sin\theta}{\theta^3}\omega_{\times R}^2 \tag{2.77}$$

$$\text{with } \theta = \sqrt{\alpha^2+\beta^2+\gamma^2}$$

$$exp(\boldsymbol{\omega}_\times) = \left(\begin{array}{c|c} \exp(\boldsymbol{\omega}_{\times R}) & \boldsymbol{V}\mathbf{t} \\ \hline 0 & 1 \end{array}\right) = \left(\begin{array}{c|c} \boldsymbol{R} & \mathbf{t} \\ \hline 0 & 1 \end{array}\right) = \boldsymbol{T}, \text{ with } \boldsymbol{T} \in SE(3) \tag{2.78}$$

2.3.5.4 Logarithmic map of $SE(3)$

Similar to SE(2), the inverse $SO(3)$ rotation part of a group element can be calculated separately as shown in Section 2.3.3.5, and $\boldsymbol{V}$ can also be inverted (cf. [90] Eq. 9.26) to perform the logarithmic mapping:

$$\ln\left(\begin{array}{c|c} \boldsymbol{R} & \mathbf{t} \\ \hline 0 & 1 \end{array}\right) = \left(\begin{array}{c|c} \ln(\boldsymbol{R}) & \boldsymbol{V}^{-1}\mathbf{t} \\ \hline 0 & 0 \end{array}\right) = \boldsymbol{\omega}_\times, \text{ with } \boldsymbol{\omega}_\times \in se(3) \tag{2.80}$$

$$\boldsymbol{V}^{-1} = \boldsymbol{I}_3 - \frac{1}{2}\omega_{\times R} + \frac{1}{\theta^2}\left(1 - \frac{\theta\cos\frac{\theta}{2}}{2\sin\frac{\theta}{2}}\right)\omega^2_{\times R} \tag{2.79}$$

$$\text{with } \theta = \sqrt{\alpha^2+\beta^2+\gamma^2}$$

2.4 2D Camera Calibration

Visual perception systems offer versatile applications for robotics and automation systems. To ensure a constant and consistent sensor data representation it is often necessary to know the diverse physical characteristics of the employed sensor system.

In common digital 2D-camera systems the employed camera optics and sensor-system define these characteristics, the so called *intrinsic camera parameters*, and can usually be modeled within a parameterized camera model like presented in bellow Section 2.4.2. Besides these intrinsic parameters, it is also important to know the spatial relation of the camera system itself to the perceived world, target, or another relative system. These relations are the *extrinsic camera parameters* and are introduced in Section 2.4.1.

2.4.1 Extrinsic Camera Parameters

Extrinsic parameters describe the spatial position of a camera within any known reference system. In a common 3D coordinate space such a connection can be simply described by its position and rotation (cf. Section 2.1). Therefore, the extrinsic *pose* of the camera is represented by a homogeneous transformation matrix ($\boldsymbol{T}$) as shown in Figure 2.10.

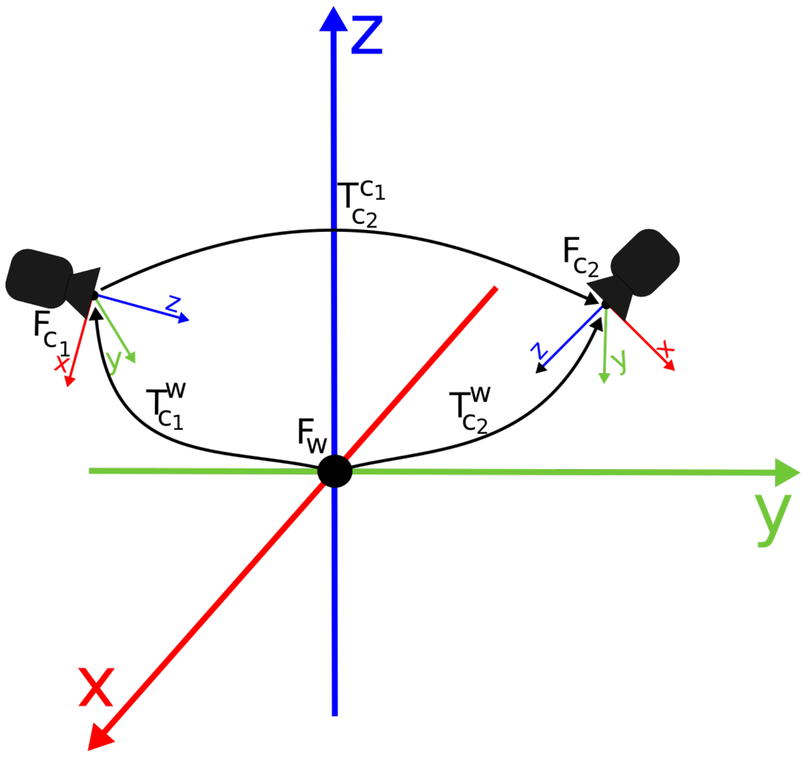


Figure 2.10: Extrinsic camera parameters. F_w: world coordinate frame, $F_{c_{1/2}}$: camera coordinate frame of a specific camera, $T^{w}_{c_{1/2}}$: Transformation between a world frame and the specific camera frame, $T^{c1}_{c_2}$: Transformation to transform information from F_{c_1} to F_{c_2}.

2.4.2 Pinhole Camera Model and Calibration Parameters

The pinhole camera model can be seen as the standard camera model used in many and versatile applications in the field of computer graphics, robotics, and CV. The parameters introduced with this model are visualized in Figure 2.11 (cf. [94]).

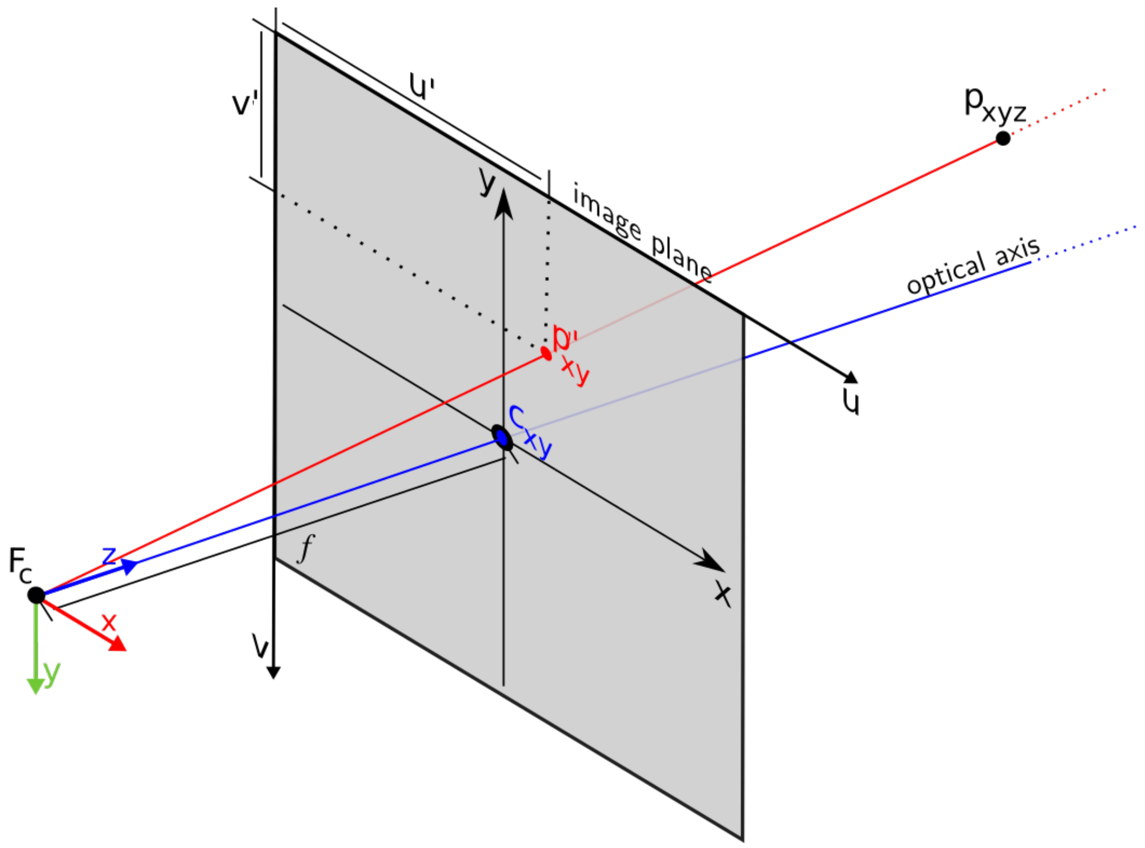


Figure 2.11: Visualization of the basic pinhole camera model and its parameters. F_c: camera coordinate system (frame), p_{xyz}: point in $\mathbb{R}^3$, p'_{xy}: point p projected into the image plane, x/y: image plane (or *principal plane*) coordinate system, u/v: specific pixel coordinates within the image plane, c_{xy}: principal point, f: focal length.

The camera frame F_c, originating from the center of the physical aperture of the camera system, describes the camera's intrinsic 3D reference system. Each observed point p_{xyz} can be projected onto a virtual image plane with the help of the focal length f and the calculated principal point, describing the origin of the image's main coordinate system. This point physically relates to the center of the image sensor (usually found by a calibration process) and the optical axis can be defined as a virtual axis starting from the sensors 3D space origin pointing to the intersection of the image plane at the principal point's 2D location (cf. Figure 2.11).

As most applications use pixel-based coordinates to organize image data in columns (u) and rows (v) the u/v coordinate system is originated on the top left corner of the image plane. The plane itself is basically identical with the x/y frame but organizes each pixel with positive pixel coordinates in the particular u and v direction.

2.4.2.1 Image Plane Projection

Assuming that all extrinsic parameters, as well as the effective sensor size s_x and s_y are known, it is possible to map each 3D point (p_{xyz}) within the camera reference frame to the associated 2D (p'_{xy}) image-plane and respective u/v-pixel coordinates through geometric relationships illustrated as follows:

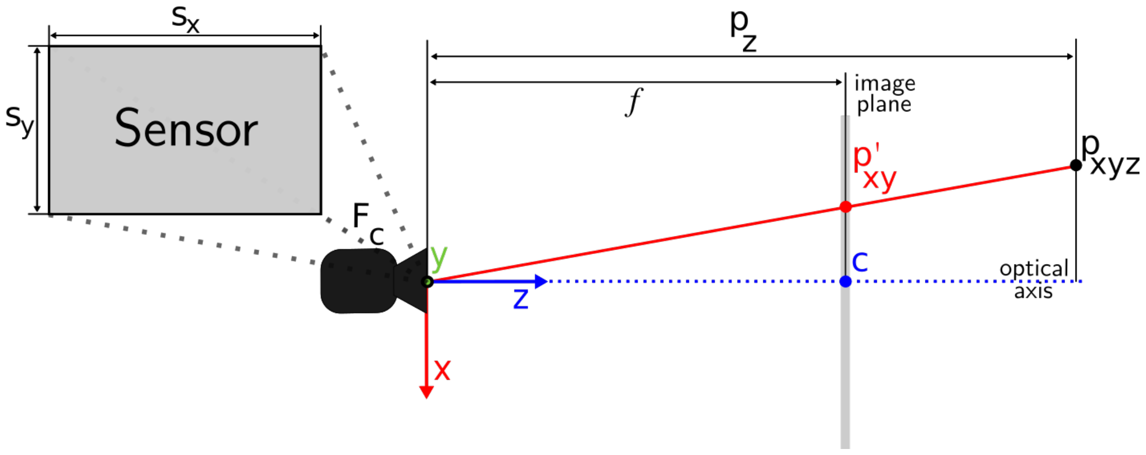


Figure 2.12: Visualization of the image plane projection. F_c: camera coordinate system (frame), p_{xyz}: point in $\mathbb{R}^3$, p'_{xy}: point p_{xyz} projected onto the image plane, c: principal point, f: focal length, s_{xy}: effective sensor size.

$$p' = \begin{pmatrix} \frac{p_x f}{p_z s_x} \\ \frac{p_y f}{p_z s_y} \end{pmatrix} \tag{2.81}$$

$$u = p'_x + c_x \tag{2.82}$$

$$v = p'_y + c_y \tag{2.83}$$

The focal length (f) characterizes the physical distance between the image plane and sensor, and the principle point (p) describes the physical center of the sensor and is defined as the position where the optical axis hits the sensor. It is also recommended to apply an additional distortion model to the point projection in order to consider the accurate optical characteristics of the camera lens. Together with the focal length and the principle point such distortion parameters forms the so called *intrinsic camera parameters* and complete the *pinhole camera model*. As the final distortion of a camera system is heavily related to its final manufacturing, and so heavily affected by mechanical tolerances, these distortion parameters are usually estimated by a calibration process [95, 96, 97, 98].

It is also common practice to organize all parameters necessary for the image plane projection in a single 3×4 matrix $\boldsymbol{P}_{proj}$. This allows to perform the image plane projection with a single matrix multiplication based on homogenized coordinates as shown in Eq. 2.85.

$$\boldsymbol{P}_{proj} = \begin{pmatrix} fs_x & 0 & c_x & 0 \\ 0 & fs_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.84}$$

$$p'_{u,v} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \boldsymbol{P}_{proj} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} \tag{2.85}$$

Note that many applications are based on different orientations of the camera base frame in relation to the image plane. This difference has to be considered when employing the here presented equations. The additional distortion correction is also applied after the described image plane projection and usually involves an additional affine transformation (cf. 2.1.4) to adjust the final pixel positions.

2.4.3 Perspective-N-Points (PnP)

The Perspective-N-Points (PnP) problem is a renown task in CV that gained much research attention in the last decades and affects many domains like extrinsic camera calibration or object detection [99]. The problem occurs during estimation of the 3D position of an object (cf. $\boldsymbol{T}_C^O$ in Figure 2.13) within the camera's coordinate-space (F_C) based on N known object keypoints (p_n) within the 3D intrinsic object coordinate system (F_O) and their corresponding 2D projection (p'_n) on the image plane as illustrated in the following Figure 2.13.

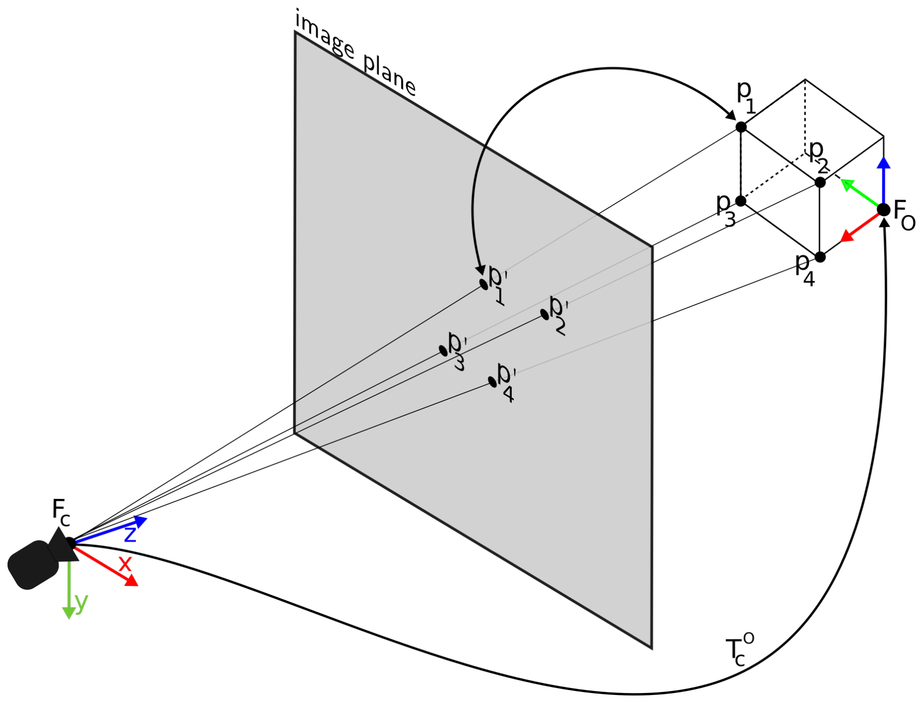


Figure 2.13: Illustration of the PnP problem. The projection of the 3D object points (p_{1-4}) lead to a 2D point pattern (p'_{1-4}) on the image plane. Solving a given PnP constellation will estimate the transformation ($\boldsymbol{T}_C^O$) between the observing camera coordinate frame (F_C) and the respective object frame (F_O).

Now, it is literally a *question of the perspective*. The loss of the third dimension while projecting the estimated object points on the image plane introduces an ambiguity as multiple transformations between F_C and F_O could lead to a given projection pattern. A solution to the PnP problem is provided by the transformation $\boldsymbol{T}_C^O$ that fits the given point correspondences which theoretically can be found when at least three unambiguous point correspondences are known. As in many estimation tasks, the given correspondences should always be as accurate as possible. Heavy outliers but also minor inaccuracies in the data will always have a negative impact on the estimation result and should be considered in a final system design [100].

Many approaches have been published that try to tackle the problem iteratively. Quan *et al.* [101] solved the problem by redundantly fitting linear equation systems for each point correspondence which need at least four correspondences to solve the given three unknowns. More data-focused algorithms [102] employ an additional Gaussian noise model to take care of a probably high noise of the given corresponding point-pairs. As such, iterative solutions can be seen as computationally greedy algorithms, also non-iterative approaches that try to provide a closed-form solution for the problem are also available in the literature. Lepetit *et al.* published a well-employed method, called *EPnP* [103] that needs at least four correspondences and formally describe the problem as a weighted sum of four imaginary control points. These control points are adjusted based on quadratic optimization and the given point correspondences. The *EPnP* algorithm also serves as the backbone for the autonomous extrinsic calibration procedure based on *RoPose* as detailed in Section 3.4.

It is obvious that non-iterative methods are perfectly applicable when facing large sets of correspondences and that they could save a lot of computation time. As shown by Ferraz *et al.* [104] an additional rejection of possible outliers can be used to improve the performance of the algorithm. Outlier rejection especially becomes helpful when the observed keypoint positions (in both domains, 2D and 3D) are also suffering from higher detection errors and could be performed based on well-known methods like Random Sample Consensus (RANSAC) [105].

The classical PnP-problem has long been in the focus of several research groups. More modern solutions are available which were usually developed for very specific applications and domains and should not be seen as generally solutions but may worth consideration if a problem at hand is comparable to a given solution [106, 107, 108, 109].

2.5 Optical Flow and Motion Estimation

In general, the term *optical flow* describes the time related movement representation of a known element based on a well defined supervising perspective. In CV it is usually employed to describe the effect of a distinct movement on a given image. Based on the formal optical flow definition in Eq. 2.86 (cf. [110]), the optical flow was originally defined directly on the pixel-level based on their dedicated positions (x,y), the given intensities (I), and a defined time base (t).

$$I_{t+1}(x,y,t) = I_t(x,y,t) + \frac{\partial I_t}{\partial x}dx + \frac{\partial I_t}{\partial y}dy + \frac{\partial I_t}{\partial t}dt \tag{2.86}$$

$$I_{t+1} = I_t + \Delta x + \Delta y + \Delta t \tag{2.87}$$

The given partial derivations describe the time-related intensity shift. This intensity-based view allows for a temporal direction aware tracking of pixel movements. The approach is based on principles of fluid dynamics and can be transferred to model abstract movements, e.g. of estimated objects, joints, or every imaginable kind of instance or agent, in general. The original authors Horn and Schunck distinguish between the optical flow itself, which describes the intensity flow of the holistic image and the motion field, introducing additional constraints and thus focusing on the projections of 3D motion trajectories [111]. When describing such 2D motions of an object, the formal definition can be transferred to describe the $t+1$ position of a geometric primitive.

$$P_{t+1}(x,y) = P(x_t + \Delta x, y_t + \Delta y) \tag{2.88}$$

Eq. 2.88 can be used to describe the motion of such an primitive object from one observation to another in the form of a normalized direction vector ($\mathbf{m}$), the transformation distance (d), and also the speed (s) related to the given timebase.

$$d = |P_{t+1} - P_t| = \sqrt{\Delta x^2 + \Delta y^2} \tag{2.89}$$

$$\mathbf{m} = \frac{1}{d} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \tag{2.90}$$

$$s = \frac{d}{\Delta t} \tag{2.91}$$

Usually, despite the original author's point of view (cf. [111]), the term *optical flow* is often used synonymously with a geometric based motion description formally called *motion flow*.

2.6 Neural Networks

“A deep-learning system doesn’t have any explanatory power.”

Geoffrey Hinton, 2017 - [112]

Neural networks, especially convolutional neural networks build the backbone of modern State of the Art (SOTA) computer vision applications, just as the here presented systems and approaches. The following sections provide an overview of some parts, elements and building blocks that are needed to build such networks and realize various applications.

2.6.1 Artificial Neurons and Networks

An artificial neuron is in effect a single response-unit with tweakable or trainable scalar weight factors (w_n). It is responsible for a certain output (y) according to a single or potentially multiple inputs (x_n), and an optional trainable bias therm (c_{bias}) [113]. A common model of such an artificial neuron is shown in Figure 2.14.

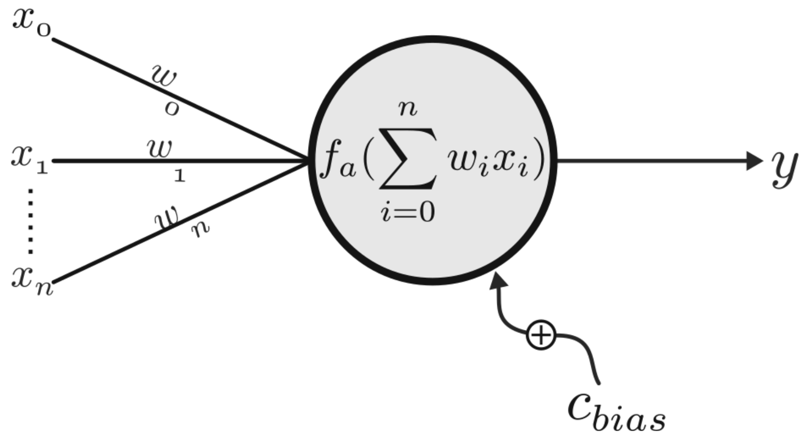
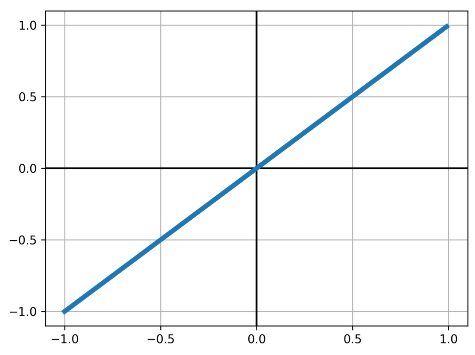
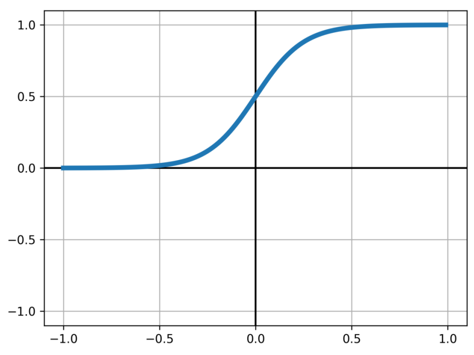


Figure 2.14: Single neuron and its parameters. Each neuron can have several inputs (x_n) which are individually weighted by a trained weight (w_n). The weighted sum of all inputs form the parameter for the activation function (f_a) which is used to calculate a neurons final output. An optional but also trained and optimized bias term (c_{bias}) can be used to finally affect the output (y) which will increase or reduce the global influence of a single neuron’s output (cf. [113]).

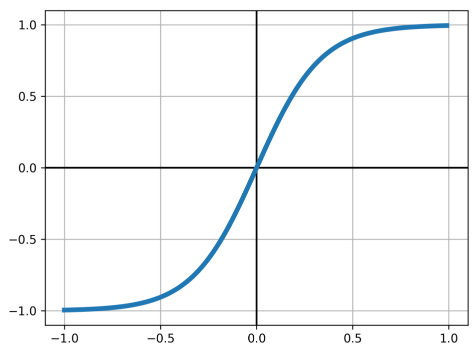
Each unit internally accumulates the weighted inputs and passes the result into a chosen activation function (f_a), which will finally lead to the neuron's response. Just as the adjusted weights, the activation function has a significant impact on y. Figure 2.15 provides a simplified overview of possible and commonly employed activation functions.



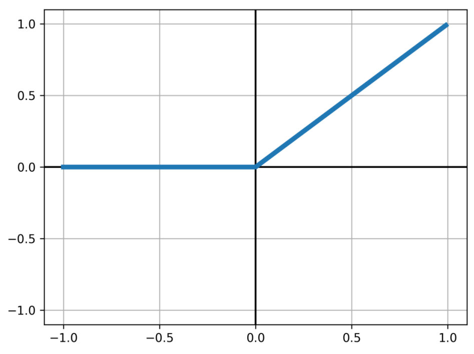
(a) Linear - $Lin()$
$f(x) = x$



(b) Sigmoid - $Sig()$
$f(x) = \frac{1}{1+e^{-x}}$



(c) Hyperbolic tangent - $TanH()$
$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



(d) Rectified linear units - $ReLU()$

$$f(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

Figure 2.15: Overview of common activation functions, usually with an normalized output value range between -1.0 to 1.0. (a) Linear activation functions add no additional influence to the weighted inputs. Usually employed for the last stage of a regression network and also for input neurons. (b) Sigmoid activation functions wrap the output in a continuous asymptotic scale between 0.0 and 1.0. (c) Hyperbolic tangent activations are very similar to sigmoid functions but scales between -1.0 and 1.0. (d) A Rectified Linear Units (ReLU) are probably the most popular activation function for hidden layers. ReLU suppress negative outputs and employ a standard linear response for $x > 0$. Equations are based on the publication from Nwankpa *et al.* [114].

0

Besides the introduced activation functions in Fig. 2.15, modern applications offer many more advanced methods like *LeakyReLU* [115] which adds an additional independent linear response for $x < 0$ to the standard *ReLU*, or *SoftMax* based functions usually employed to solve classification tasks. A more broad and also formal overview of the different activation functions and their final effects, drawbacks and variants was previously published by Ramachandran *et al.* [116] and Virtanen *et al.* [117].

To finally form a NN, the artificial neurons need to be combined in connected constructs of various size. Each NN can be described as a structured and usually multi-layered combination of multiple single artificial neurons as shown in Figure 2.16, each layer between the in- and output structure is called a hidden layer.

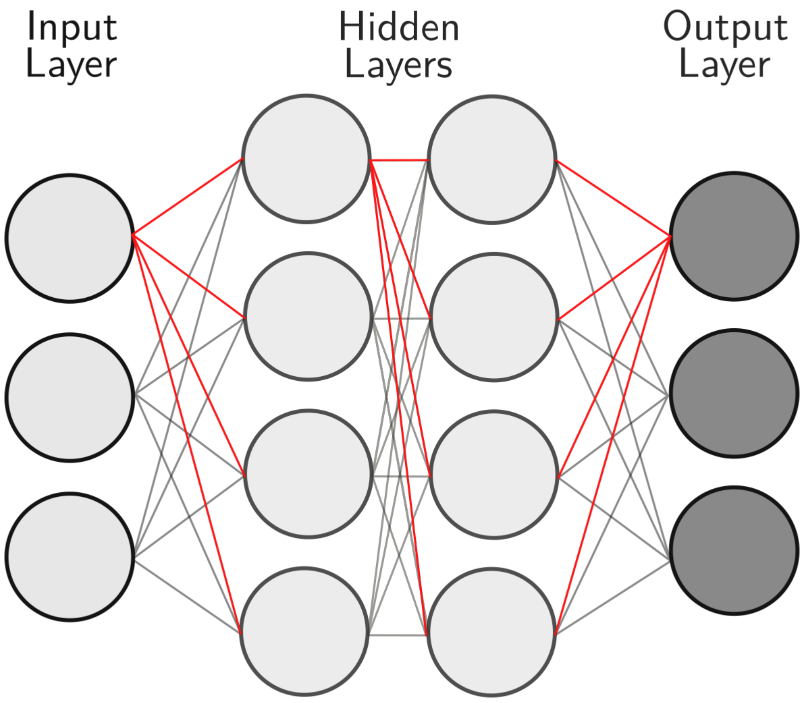


Figure 2.16: Multiple connected neurons forming a NN. This example is called a dense, fully connected NN where all neurons from the particular direct neighboring layers are connected.

All neurons of a resulting NN and their respective weights can then be optimized to minimize a given loss, which can be extracted from the network's output and a given Ground Truth (GT) (cf. Section 2.6.3).

When talking about deep learning, the so called Substantial Credit Assignment Path (CAP) can be used to describe the depth of a neural network as a scalar value and it is defined by counting all the given hidden layers of a network plus the final output layer (cf. [118]). The common opinion is that the term *deep learning* simply describes an application of a NN with more than one hidden layer ($CAP > 2$).

This thesis does not provide an extensive formal introduction to the basic math exploited in NNs as it has been sufficiently reviewed previously. For example, Goodfellow *et al.* covered all necessary principles in the first part of their open-source book on deep learning [119].

2.6.2 Convolutional Neural Networks

A CNN can be seen as an uncompromising transfer of a neural network learning structure to spatial convolutions. As convolutions have emerged from many very physically embossed applications like analog and digital signal processing (cf. [120]), this operation has become one of the most employed tools in the field of CV and forms the basis for many popular basic operations like blurring, smoothing or sharpening (cf. [121]), and also advanced operators such as the commonly used *Canny edge detector* [122]. Convolutions in CV are using 2D filter kernels with a defined size $n \times n$ and an associated weight mask as visualized in Figure 2.17.

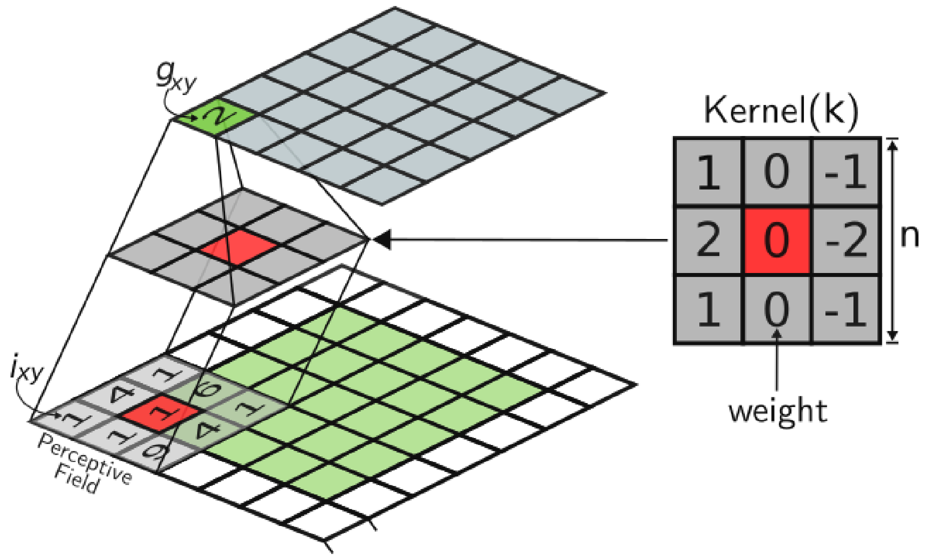


Figure 2.17: Visualization of a spatial convolution operation on 2D input data. The $n \times n$ weight matrix, the so called filter kernel, contains the weights which are applied on each input element within the kernels receptive field. The final filter response can then be calculated by sliding the filter kernel across the whole input image and summate the weighted input elements for each kernel appliance.

The convolution itself is performed by sliding the convolution matrix (k) through each pixel of a 2D input and applying the $m \times n$ sized filter mask's weights of a usually squared filter to the particular input intensity (basically an element-wise multiplication) in the considered neighborhood, usually called the receptive field. The final filter response

(g_{xy}) is the accumulation of the weighted (each weight is indicated by k_{xy}) inputs (i_{xy}) transferred to the current position of the filters base element (red in Figure 2.17). The filter-matrix itself is usually flipped. This, often also called neighborhood operation, is formally defined by Eq. 2.92 (cf. [119]).

$$g(x,y) = i * k(x,y) = \sum_m \sum_n i_{xy} k_{x-m,y-n} \tag{2.92}$$

In the difference to standard NNs, the learning process in CNNs will affect the weights of multiple convolution matrices within the network structure and thus allows for the learning of spatial correlations of the given input. In practice, such convolutions are also affected by additional parameters to tweak the final filter response. Each of the following options will affect the output size of the response and have to be considered when defining such network structures.

- **Kernel Size**
  The kernel size ($s_k \in \mathbb{N}$) simply describes the number of weights a kernel has on each side (e.g. 3×3 as shown in the here presented examples).

- **Stride**
  The stride ($stride \geq 1, stride \in \mathbb{N}$) describes the amount of movement between each filter application. It can basically be used to skip input elements and introduce a downsampling-like behaviour.

- **Input and Output Padding**
  Padding is used to adjust the final response dimensions of filter operations. Depending on the kernel and input sizes the final response may end up in lower dimensions. To avoid this scenario, padding elements are added before or after the performed convolutions. An example of input padding is indicated by the white border in Figure 2.17. Usually, the padding elements are set to zero (*Zero-Padding*).

- **Dilation**
  A dilation factor ($dilation \geq 1, dilation \in \mathbb{N}$) defines the spacing between each filter weight. A dilation of 1 is therefore the standard convolution kernel. As described by Yu *et al.* in their original work [123], dilated convolutions can drastically increase the receptive field and may help to aggregate information on various scale factors by also keeping the filter's dimensions in controllable sizes.

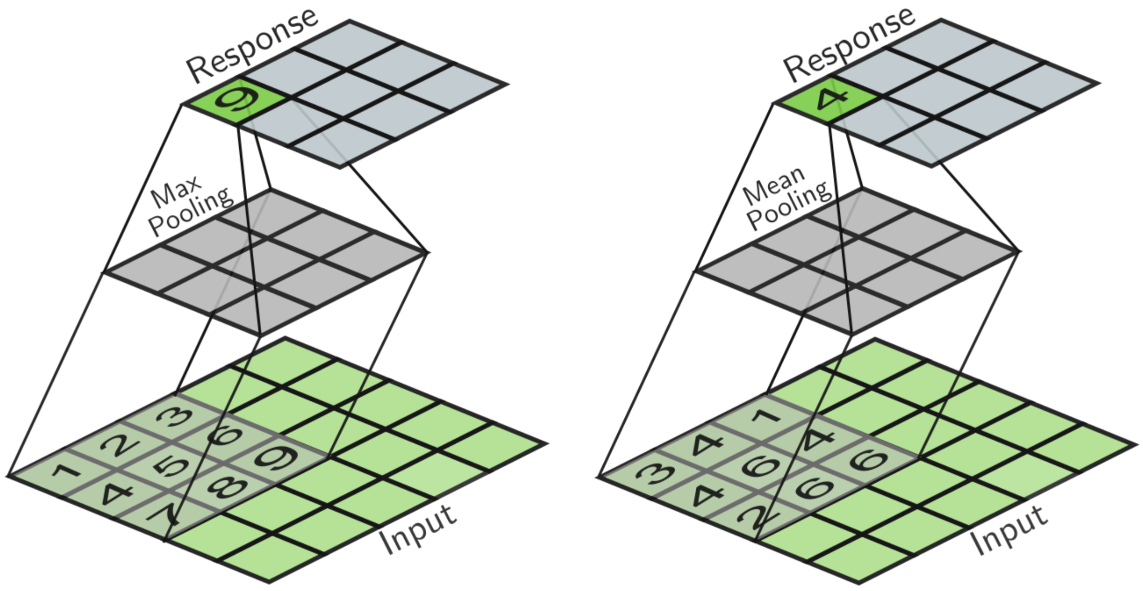
As a formal introduction of all these options and their final impact lies way beyond the scope of this thesis, it is referred to Dumoulin and Visin's comprehensive introduction [124][1], or the documentation of one of the various publicly available deep learning frameworks like *Tensorflow*[125], *Pytorch*[126] or *CNTK*[127].

2.6.2.1 Additional Building Blocks

Besides the standard and most common convolutional layers, CNN architectures can contain additional elements and building blocks that would fundamentally affect the extracted features and the final output.

Pooling Layer

Just as standard convolutions, pooling layers are also spatial operators and are used to reduce the input dimensions. Figure 2.18 visualizes the effect of the common max and mean pooling operators used in CNN architectures.



(a) max pooling operation (b) mean pooling operation

Figure 2.18: Visualization of max- and mean-pooling operations. (a) The response of a max-pooling operation is simply the maximum value within the receptive field. (b) Mean-pooling operation calculates the mean of all elements within the kernel's receptive field.

While a max-pooling layer's response is simply the maximum value within the receptive field, mean-pooling operations calculate the mean value. The pooling is applied across the whole input and thus can also be adjusted by options like stride and padding while the chosen kernel size mainly affects the response's dimensions.

[1]In the associated repository you can also find their easily understandable animations `https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md`

Transposed Convolutional Layers

A transposed convolutional layer performs the transposed operation of a convolutional layer. Thus they are often also called *de-convolutional* layers. Such layers are used to perform a parameter based, feature extracting upsampling, by exploiting the trainable parameters of a filter mask. An example of a de-convolution is visualized in Figure 2.19.

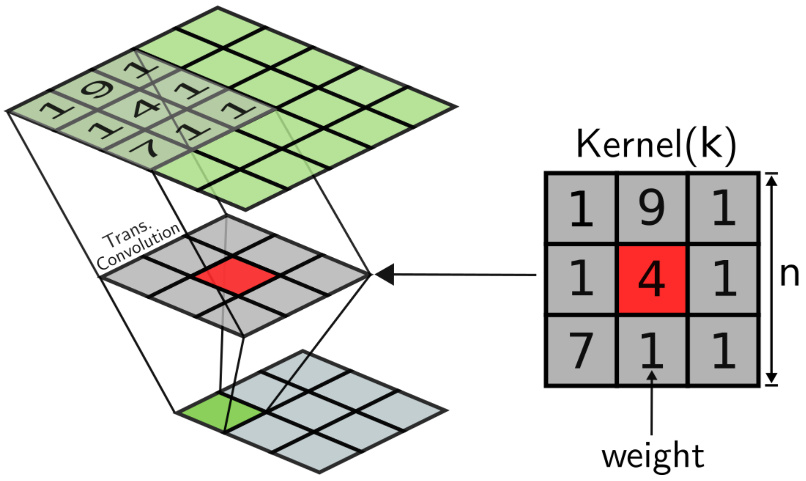


Figure 2.19: Visualization of a de-convolutional operation on 2D input data. The trainable weight matrix is used to increase the dimensions of the final response according to the given parameters like kernel size and stride.

Batch Normalization

A technique called batch normalization has been published by Ioffe *et al.* in 2015 [128] and helped many applications to reach new standards. As normalization is a crucial part of every machine learning application and traditionally applied to in- and output data, normalization methods like batch normalization are directly injected as layers into the network structure itself. It should be seen more as a sort of transformation than an extra layer. They were designed to reduce the so called *covariate shift* between mini batches. The covariate shift describes the differences of the distribution in the final neurons response between optimization steps. Because new data is processed within each training-step, and each training-step itself optimizes the weights of each neuron a heavy distribution between each batch can lead to a heavy impact on the neurons weights adjustments during optimization especially in high level network layers. Although this is the majority opinion, Bjorck *et al.* [129] claims that it is more likely the *smoothening effect* of this normalization that is responsible for its positive effects which is also a reasonable explanation for the success of this method. However, this transformation technique can speed up and stabilize training routines as well as optimizations by normalizing each output within the network based on the mean and standard deviation for each mini-batch (cf. Section 2.6.3) used for training.

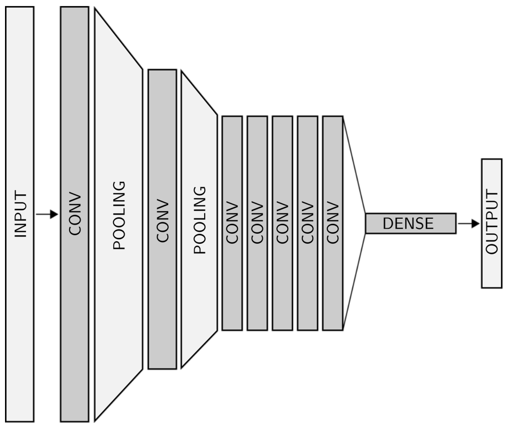
Fully Connected Layer

Fully connected layers were already described in Figure 2.16 and describe specific layers in which each neuron is connected to each neuron of the previous layer. Within the multidimensional layer structure of a CNN, this principle is simply transferred to this layer structure. Fully connected layers can be used to perform a network's final regression e.g. to a class probability. As every input of a neuron (cf. Figure 2.14) will increase the parameter size of the final network, such layers always have a massive impact on the model size which is a major problem especially in embedded applications, that has to be considered when designing new networks.

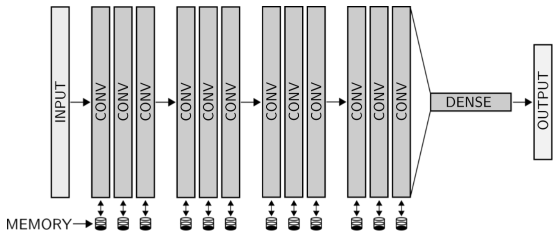
2.6.2.2 Network Designs

There are literally no restrictions when it comes to defining a NN or CNN structure, and describing all available base architectures would be far beyond the scope of this thesis. In order to provide a basic understanding of network design and common practices the example network architectures summarized below in Figure 2.20 will be thoroughly discussed.

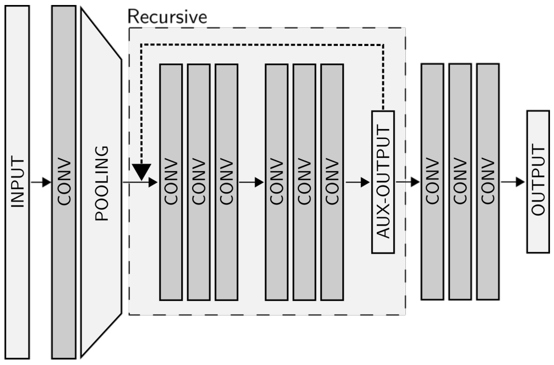
A standard *feed-forward network* as outlined in Figure 2.20 (a) is the most straight forward CNN structure. The input is simply propagated through all network layers and the results can be obtained at the output element. Such networks, e.g. the the *VGG*-networks from the famous Visual Geometry Group (VGG) [130], often find their usage in feature extraction tasks and so build the backbone for many applications (e.g. [I, II]).



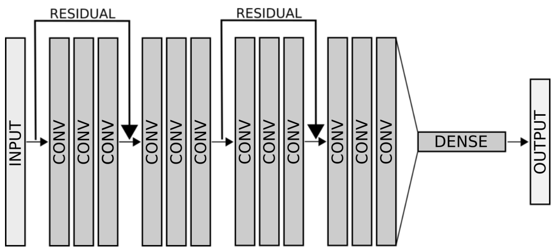
(a) Feed Forward Network



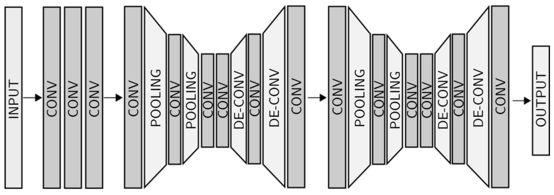
(b) Recurrent Network



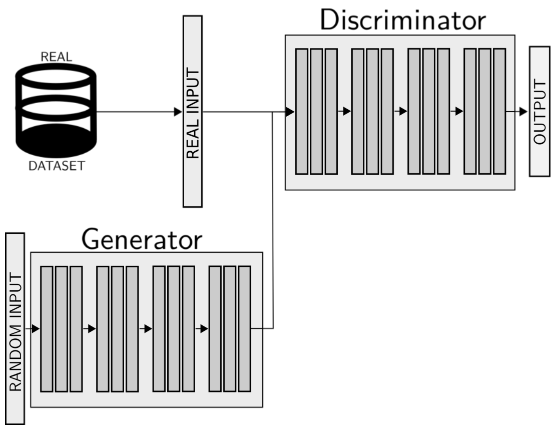
(c) Recursive Networks



(d) Residual Networks



(e) Stacked Hourglass Network



(f) Generative Adversarial Network

Figure 2.20: Examples of common network architectures and structures. (a) A feed-forward network forms the most basic, linear structure of a CNN. (b) Recurrent networks are able to preserve information from previous network states and so enable access to some kind of history. (c) Recursive networks consist of repetitive sub-network structures based on the same in- and output model to increase the network's overall capacity. (d) Residual networks allow to skip some layers of the network by introducing adaptive shortcuts. (e) Stacked hourglass structures consist of multiple convolution and de-convolution stacks to allow for a feature extraction on multiple levels. (f) A Generative Adversarial Network (GAN) is a compound network structure of two competing networks where the *generator* part tries to generate artificial data elements as realistic as possible while the discriminator judges the *realness* based on given real data examples.

In contrast to a standard feed-forward network where each propagation is limited to its intrinsic knowledge, *recurrent networks* are able to preserve information from previous states (see Figure 2.20 (b)). This enables new possibilities for every task that is somehow dependent on continuous information from previous states, for instance speech- [131, 132] or action recognition [4, 133]. The history of such networks can be traced back to 1997 [134] when Hochreiter *et al.* proposed the famous and still heavily used Long Short-Term Memory (LSTM) networks that evolved into slightly more simple variants just as *Gated Recurrent Units* [135].

As the term *recursive* indicates, *recursive networks* are based on multiple repetitions of whole network parts as shown in Figure 2.20 (c). The fact that the input and output structure of the repetitive parts are consistent allows for a theoretically infinitely large or deep network. However, it usually does not make any sense to lift a network's dimension above a reasonable scale as detailed by Stathakis *et al.* [136].

Residual networks are also very similar to standard feed-forward networks but introduce the possibility to skip particular layers by defining adaptive *shortcuts* (see Figure 2.20 (d)). This reduces a network's final complexity especially when it comes to very deep structures. The possibility to eventually skip parts of the network mainly reduces the training and inference time and also counters the problem of vanishing gradients (cf. Section 2.6.3). Such a procedure also does not change the basic behavior of the defined network because the neurons of the skipped network-parts are just ignored within the feed-forward operation. The *ResNet* structures [137], which are commonly used for image feature extraction, are one of the most popular representatives of this kind of network.

Stacked hourglass networks have emerged in 2016 [138] and have their origin in human pose estimation. The origin of the architecture's name becomes clear when looking at the sketch in Figure 2.20 (e). Such a network's main building blocks are multiple combinations of stacked convolutional and de-convolutional layers, which leads to the eponymous shape. Such a structure allows for a multi-scale feature extraction depending on the convolutions and de-convolutional layers of each hourglass stage.

In 2014, Ian J. Goodfellow *et al.* proposed the GAN [139]. The idea behind GANs is the competition of two NNs against each other. As outlined in Figure 2.20 (f), the generator network produces artificial data (e.g. images like in most CNN applications) out of a randomized input. The so called discriminator network has access to real data examples and predicts a probability for the artificial generated data's *realness*. This will lead to the mentioned competition where the generator will be trained to generate data as real as possible to *fool* the discriminator. GANs are enjoying great popularity and it is obvious that their self-learning behavior, mainly introduced by the generator, can be used to realize interesting applications for a wide range of un-, semi- and self-supervised learning structures especially because the generator's and discriminator's architecture is also not limited to a specific application domain and can be adjusted to tackle various tasks.

All of the here presented architectures can usually be exploited in a rather wide field of application and can be combined and extended with multiple inputs, outputs, losses, and other networks to fulfill their dedicated task. However the here presented work mainly utilizes standard feed forward (cf. Chapter 3) and recursive networks [I] but also relies on ideas from other architectures.

2.6.3 Training, optimization and Loss Functions

When it comes to the training of NNs, optimization methods are used to determine the set of parameters and weights supposed to be the best for the designed network and the given data basis. An optimization step tries to adjust the network's weight parameter constellation to minimize the resulting *losses*[2] for each processed example respectively an example-batch. Example- or mini-batches describe the amount of training data that is passed through the network in one iteration. Usually, after each epoch[3] the current parameter set will be tested using a specific test dataset that should not contain data of the training set.

Optimization is a mathematical research field itself that is way older than modern machine learning. Thus, it is again beyond the scope of this thesis to provide and extensive overview and this Section will focus on an introduction of optimization techniques and methods that were used to develop the systems presented in this work. It is referred to [140, 119] for a broad overview close to the machine learning domain.

Common optimization algorithms for NNs and CNNs can roughly be fielded with the derivative order they operate on. The so called *gradient descent optimizers* are supposedly the most naive approach, but still, one of the most used optimization methods

[2]A loss, function often also called *cost function*, is used to quantify the difference between the expected and the actual output of a network.

[3]An epoch is finished when all available training data was used once, or when a specific amount of iterations where processed if the set size is not specifically known e.g. for GANs.

even for large-scale machine learning applications [141]. They are often consolidated as Stochastic Gradient Descent (SGD) algorithms. SGD algorithms employ the gradient, and thus the first-order derivatives, of the objective or loss function (L) to update each trainable weight (w^n) provided by a NN at each mini-batch iteration t influenced by a vague set learning rate (r_{learn}) and the calculated loss-gradient ($\nabla L(...)$) extracted from the batch response (y^r_t) and the corresponding GT (y^{gt}_t). The formal equation can be given as follows:

$$w^n_{t+1} = w^n_t - r_{learn} \nabla L_t(w^n_t, y^r_t, y^{gt}_t) \tag{2.93}$$

A process called *back propagation* is used to calculate the gradients by iterating backward through the layers in the neural network, based on the extracted loss after feeding a given input to the network. Because the calculations for back propagation process can get increasingly complicated in deeper networks (the main reason for this is the necessity of employing the *chain rule* because the results of higher-level neurons normally depend on every parameter on lower network levels). The mathematical background related to back propagation is covered in more depth in [142, 143]. In addition to first-order algorithms, higher-order derivatives can also be used to update the network's parameters. Ruder *et al.* provides a broad overview of gradient-based optimization methods [144].

The loss function itself can be everything imaginable that will describe a difference between the known GT (y^{gt}) and the given response (y^r) of a network. One of the most commonly used losses is the Mean Squared Error (MSE), which could be applied to almost all vector, matrix, or tensor-based linear output formats and is formally defined by the following Eq. 2.94 where the index i denotes a single element of the output of the network or the available GT:

$$MSE(y^{gt}, y^r) = \frac{1}{n} \sum_{i=1}^{n} (y^{gt}_i - y^r_i)^2 \tag{2.94}$$

The MSE is also often extended with an additional square root calculation. The resulting Root Mean Squared Error (RMSE) also serves as a common loss function, suitable for various applications.

$$RMSE(y^{gt}, y^r) = \sqrt{MSE(y^{gt}, y^r)} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y^{gt}_i - y^r_i)^2} \tag{2.95}$$

For classification tasks, where a network's output is usually given by a vector or tensor containing probabilities for each dedicated class (c) of all known classes (C), the cross-entropy (H) is usually employed for the loss calculation:

$$H(y^{gt}, y^{r}) = -\sum_{c=1}^{C} y_c^{gt} \ln(y_c^{r}) \tag{2.96}$$

Besides a standard single loss calculation, losses can also be a, usually weighted, combination of multiple different losses. Recent publications show, that style losses where an additional loss term is calculated based on various level feature responses of the output and the GT, calculated based on an additional employed image feature extraction network (cf. Section 2.6.2.2), can be used to enable interesting new prospects, especially for CNN applications [145].

Standard SGD algorithms can be extended with adaptive optimization techniques. These methods were developed to allow for an autonomous adaption of learning rates and other parameters like biases. One of the most prominent and widely used optimization technique is called *Adam* ($\sim$Adaptive Moment Estimation)[146] and combines the advantages of a per parameter learning rate (SGD usually handles just one learning rate for all parameters) of *AdaGrad* ($\sim$Adaptive Gradient Algorithm)[147] and the more excessive adaptive *lr* adjustment of *RMSProp*[148] (Root Mean Square Propagation). *Adam* adjusts each single learning rate based on the mean and variance of the gradients, which usually helps to reduce the loss faster than standard methods.

The concept behind adaptive optimization methods like *Adam* may sound groundbreaking and ideal at first but such methods should not be seen as the sole solution. Wilson *et al.* highlighted such algorithms primarily help with generative models which are usually not facing direct optimization problems and SGD based methods with a carefully tuned parameter setting should be preferred for other problems in order to find a more optimal model [149]. The work this thesis describes put this advice into practice. The here presented learning-based systems and models were developed and debugged based on adaptive methods like *Adam* and were finally trained with SGD after ensuring a model's base functionality. In the eyes of this thesis' author, it is impossible to provide a *one fits all* solution or plan for optimization and training. At this point, it should be clear that the success of training a however designed NN hardly depends on many factors, including the employed data structure, the networks architecture and size, the supposed output, the employed normalization, the optimization techniques, and may also require a measure of intuition by the developing scientist.

2.7 Accuracy Metrics

Whenever a system needs to be evaluated or compared to previous versions and other methods, proper metrics have to be defined to allow objective performance quantification. Especially when deep learning architectures are involved the accuracy differences between various methods can be very small and a well-defined metric helps to choose the right approach for a given problem. The following section summarizes and defines the metrics that are used within this thesis. The individual metrics are used to determine the system's accuracy for a specified sub-task. Whenever this task is a classification, detection, or any other binary decision, the evaluation results can also be expressed with the precision, recall, or accuracy metrics as described in Section 2.7.1.

2.7.1 Precision and Recall

Besides an absolute metric, *Precision* (P) and *Recall* (R) calculations are used to determine the overall performance of an estimation system by rating the system's performance between 0.0 and 1.0. In order to calculate these metrics, each estimation will have to be categorized into the following categories:

- **True Positive (TP)**
  An estimation is counted as TP when the estimation system successfully detects an instance of the same class or joint as in the given GT and the similarity of the final positional estimation is higher or equal than a provided threshold.

- **False Positive (FP)**
  An estimation is counted as FP when the estimation system wrongly detects an instance of a class or joint that is not in the given GT and the similarity of a final positional estimation is higher than a provided threshold.

- **True Negative (TN)**
  An estimation is counted as TN when the estimation system does not detect a class or joint that is not in the given GT.

- **False Negative (FN)**
  An estimation is counted as FN when the estimation system does not detect a class or joint that certainly is defined in the given GT.

Based on these classifications the *Precision* and *Recall* (also called *True Positive Rate*) can be calculated as follows (cf. [150] Chap. 9):

$$Precision = P = \frac{TP}{TP+FP} \tag{2.97}$$

$$Recall = R = \frac{TP}{TP+FN} \tag{2.98}$$

The above Equations show, that a higher P, indicates less mistakenly detected instances and a higher R indicates less mistakenly suppressed instances.

To transfer the in 2.7.2 and 2.7.3 introduced absolute metrics to the P and R ratings, an additional threshold needs to be introduced. After calculating the absolute similarity between a target and an estimation, a minimum value is defined to rate if a similarity is good enough to count as a successful detection or not what finally allows for a classification of each estimation according to TP, FP, TN and FN. This procedure can be seen as common practice and can be applied to the here presented metrics by defining a minimum Intersection Over Union (IoU) or a minimum keypoint distance for a detection, e.g. 0.5, respectively $5pix$.

The evaluation of these rates at various similarity thresholds allows for plotting a precision-recall curve that can give a good overview of the estimator's overall performance (Figure 2.21). The final accuracy is then defined by the Area Under Curve (AUC) of the plot.

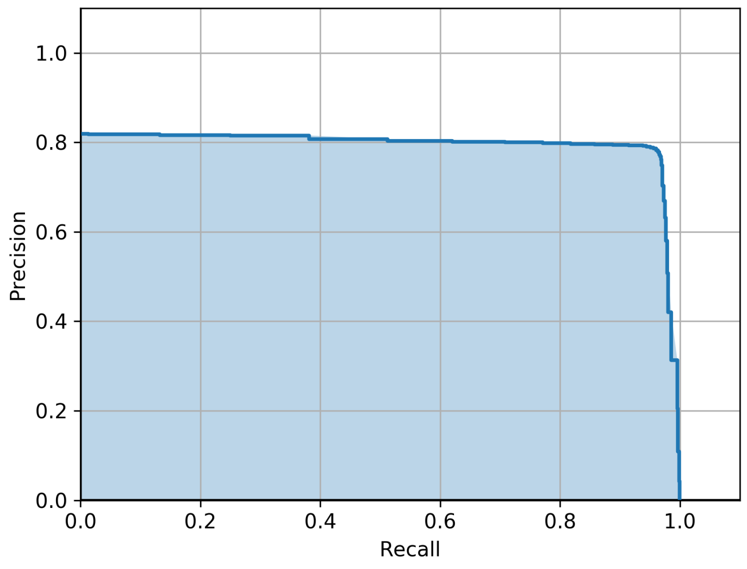


Figure 2.21: Example of a precision-recall curve. The Average Precision (AP) is defined by the Area Under Curve (AUC) (blue) of the plot.

In addition to these two indicators, the *F1-Score* can be used as a powerful single key-value e.g. for finding best models, thresholds or classifiers, by combining the precision and recall by their harmonic mean (cf. [151]):

$$F1_{Score} = 2\frac{PR}{P+R} \tag{2.99}$$

2.7.2 Bounding Box Accuracy

Bounding boxes are often used to describe a rough Region of Interest (ROI) around targets on the image plane, and are usually rectangular boxes within a known 2D space. An example of two bounding boxes (e.g. a *target*, or GT, and *prediction*) that have to be compared can be seen in Figure 2.22.

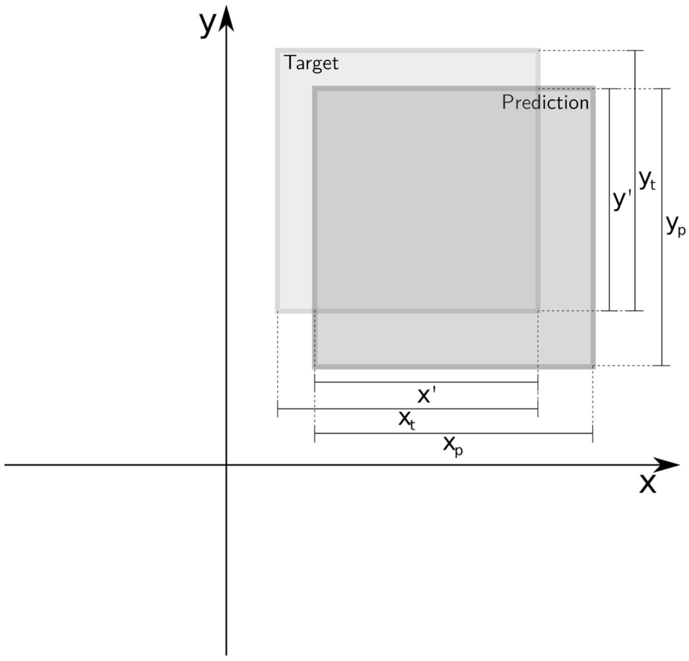


Figure 2.22: Visualization of the IoU calculation. The overlapping area is described by x' and y' and the origin areas of the bounding boxes are described by x_n and y_n.

To compute a scalar similarity factor for the two areas the so called Intersection Over Union is usually employed and can be calculated based on the combined area of both bounding boxes (A_{union}) and the area where both bounding boxes overlap (A_{over}) (cf. [152]). The respective side-lengths used to calculate the areas can also be inferred from Figure 2.22.

$$IoU = \frac{x'y'}{x_t y_t + x_p y_p - x'y'} = \frac{A_{over}}{A_{union}} \tag{2.100}$$

The IoU therefore always expresses a similarity of two 2D bounding boxes with a scalar value between 0 and 1 and thus represents a perfect metric for comparison.

Sometimes it is useful to judge the performance of a bounding box detector or a similar system by looking at various thresholds that decide if a target was successfully detected or not. In such a case the IoU can also be the basis and count each target with a IoU higher than the current threshold as good. The result is a binary classification which can be performed on a threshold range between 0.0 and 1.0. With this background eg. the F1-score (cf. Section 2.7.1) can be used as a meaningful keyvalue that leads to graphs as shown in the example in Figure 2.23.

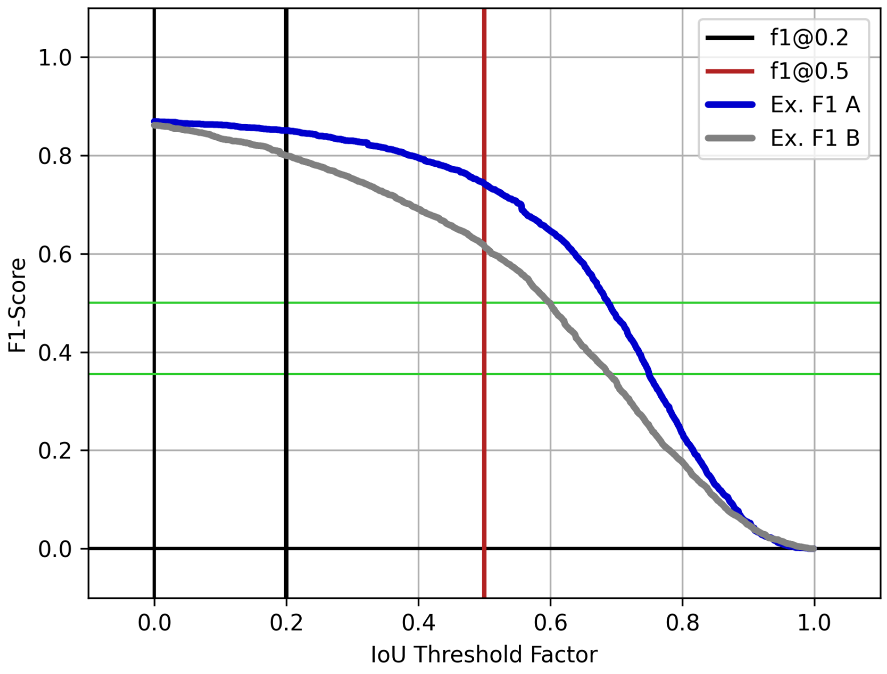


Figure 2.23: Example plot of an IoU based F1-score evaluation. Such an evaluation performs a binary classification based on various thresholds of the predicted IoUs. The graph offers a good overview of how a system performs on various accuracy levels.

Because the here presented systems do not require 3D bounding boxes, the metric used for such systems is not within the scope of this thesis. However, the presented 2D metric can be easily transferred to $\mathbb{R}^3$ by calculating an IoU based on each 3D volumes instead of 2D areas.

2.7.3 Distance and Probability of Correct Keypoints (PCK)

One of the main novelty and contribution of this thesis is the joint-wise keypoint detection of industrial robot arms. To measure the performance of such a system the standard Euclidean distances between each predicted joint position ($\mathbf{j}$) and the corresponding GT ($\mathbf{t}$) from a dataset could be sufficient to state out the absolute performance of a system. In addition, this distance will be the base for all other metrics presented below and, is calculated as follows:

$$d(\mathbf{j},\mathbf{t}) = \sqrt{\sum_{i=1}^{n}(j_i - t_i)^2}, \text{with n=2 for j,t} \in \mathbb{R}^2, \text{n=2 for j,t} \in \mathbb{R}^3 \tag{2.101}$$

Within the field of human pose estimation (cf. Section 3.3.3.1), the Probability of Correct Keypoints (PCK) metric and their possible variations are commonly used in many applications [153, 154, 138]. These metrics define a variable threshold (d_t) for the calculated distance in which a joint will be classified as correctly detected or not. The origin of PCK as published by Yang and Ramanan [155], defines d_t relative to the maximal side length of each example targets bounding box (bb_x, bb_y):

$$d_t = \alpha \max(bb_x, bb_y) \tag{2.102}$$

With the factor α $(0.0 \leq \alpha \leq 1.0)$ the absolute threshold and can be adjusted to generate meaningful plots (Figure 2.24 shows an example curve) to show how a system's accuracy changes with different thresholds, similar to the graphs detailed in Figure 2.23. According to [155], α is usually set to 0.1. To keep track of which thresholds parameter α is used, the chosen value is usually integrated into the metrics name proceeded by an @ symbol (e.g. PCK@0.1).

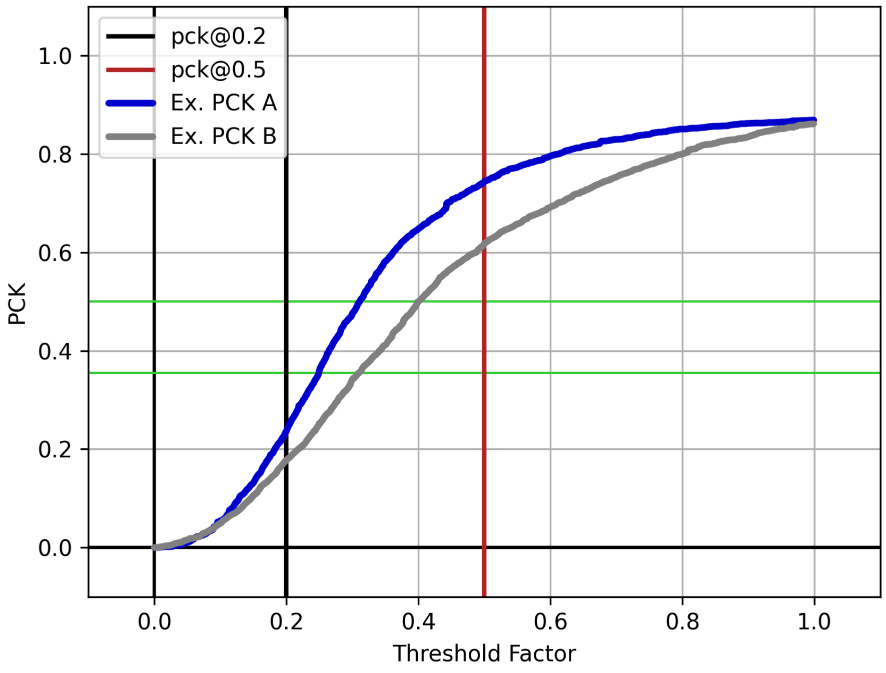


Figure 2.24: Example of a PCK plot. If the PCK based evaluation is performed for various accuracy thresholds, such a graph offers a good overview of a system's performance on various accuracy levels.

In order to create a more human-based metric, the team around Andriluka defined the *PCKh* metric in which the threshold distance is calculated relative to the length of the target human head segment and not the whole target's bounding box anymore[156]. Again, the final factor of the metric is integrated within the name (e.g. PCKh@0.5, cf. red line in Figure 2.24). Beside a plot with various factors as shwon in Fig. 2.24, the factor 0.5 is commonly used to indicate a system's performance within a *PCKh* based analysis.

3 Dynamic robot-arm supervison

3.1 Problem

As described in Section 1.2 it is important to know the current state of an industrial robot arm relative to other relevant process elements like the states of humans and objects. Such additional information enables many possibilities, e.g. to dynamically adapt the movements of the robot to the given surroundings. One highly promising approach is to employ observing vision sensors to acquire the needed and highly dynamic pose information within collaborative workspaces. Because common 2D-RGB camera sensors are becoming more and more affordable, accurate, and easy to integrate into large systems, they represent a popular source of information to extract versatile information. To also add notable value to a collaborative environment, employed smart sensory systems need to be able to extract the robot's pose information as exact and fast as possible. Relative strategies have recently been pursued excessively to estimate human poses (see e.g. Section 3.3.3.1) but their use to estimate the pose of other articulated targets like industrial robot-arms did not get the attention they deserve. The boom of data-driven algorithms has also led to huge and growing detailed labeled datasets for common use cases [156, 54, 55]. However, coming to not that popular applications, a lack of appropriate datasets for special domains is apparent.
The following part of this thesis describes how to fill this data-gap for data-driven industrial robot-arm pose estimation applications with an automated dataset generator for simulated and real robot-arms, and shows how to adapt known pose-estimation systems to the given domain.

3.2 Related Work

Many researchers, groups, engineers, and companies work in the field of *visual servoing* based on RGB-image data [157, 158, 159]. It is almost impossible to provide a wide and general state of the art overview because the use cases for such systems are as endless as the possible objects the particular solution want to observe. The focus of this thesis stays therefore within the area of industrial robotics and articulated targets.

Joint collaborative workspaces for humans and industrial robots usually rely heavily on statically defined areas [160, 161, 162] and calibrated camera systems where the extrinsic transformation between the supervising sensor system and the robot is known. In contrast to such approaches, one of the main goals of the here presented work is to estimate the actuator's kinematic pose directly from the raw image data of a observing camera system. This avoids a separate extrinsic calibration of the sensor and allows for more dynamic camera placement.

To allow for more general and target-independent solutions, a lot of work concentrates on marker-based methods. Beside known passive targets [160, 163], such as the popular *AprilTags* [10, 9], which our research group also employed in our projects as local references to allow for a more dynamic gripping process, active, e.g. infrared-based, systems can also be used to extract the pose information on an image [164]. For uncommon problems or rater uncommon observation targets of the pose estimation, Lundeen *et al.* ended up attaching visual markers even on huge excavators [165]. However, because it is one of the main goals of the here presented work to realize such an application without additional visual targets, to avoid unnecessary physical adaptions, the presented approach has to work without markers and concentrates on the bare appearance of the robot target.

To improve their vision-based control system Bohg as well as Widmaier *et al.* used pixel-wise classification [166] and random forest regression [167] based on synthetically generated data. In contrast to the aimed 2D image input, both systems operate solely on depth data which is not available in the targeted setup without putting additional effort in the sensor data processing [168, 169].

Gratal *et al.* published a possible solution by fitting a known virtual 3D model of the robot's end-effector into the acquired sensor data to estimate and track the pose [170]. As the main purpose of the proposed system is to observe the whole workspace of a robot (including all kinematic joints), estimating only the end-effector's pose is not sufficient without additional knowledge of the robot's intrinsic controller state.

In contrast to the here discussed approach for 2D joint estimation in an image, Lei *et al.* [171], Miseikis *et al.* [172, 173], Zhou *et al.* [174], as well as Heindl *et al.* [175, 176] focus on directly extracting the 3D joints out of the input image by also employing various deep learning methods. Also similar to the here presented method, Lee *et al.* [177] performed a two-dimensional heatmap regression to extract the pose keypoints of the robot system. Finally, they also employed the simple baseline [178] approach that was adopted in the here presented system.

3.3 RoPose

3.3.1 Idea

The main idea of the proposed *RoPose* system is the estimation of a 2D pose constellation of an industrial robot system in order to provide rich state information of the target. As shown in Figure 3.1 the abstract 2D position information of each robot joint needs to be extracted from a raw 2D RGB input image.

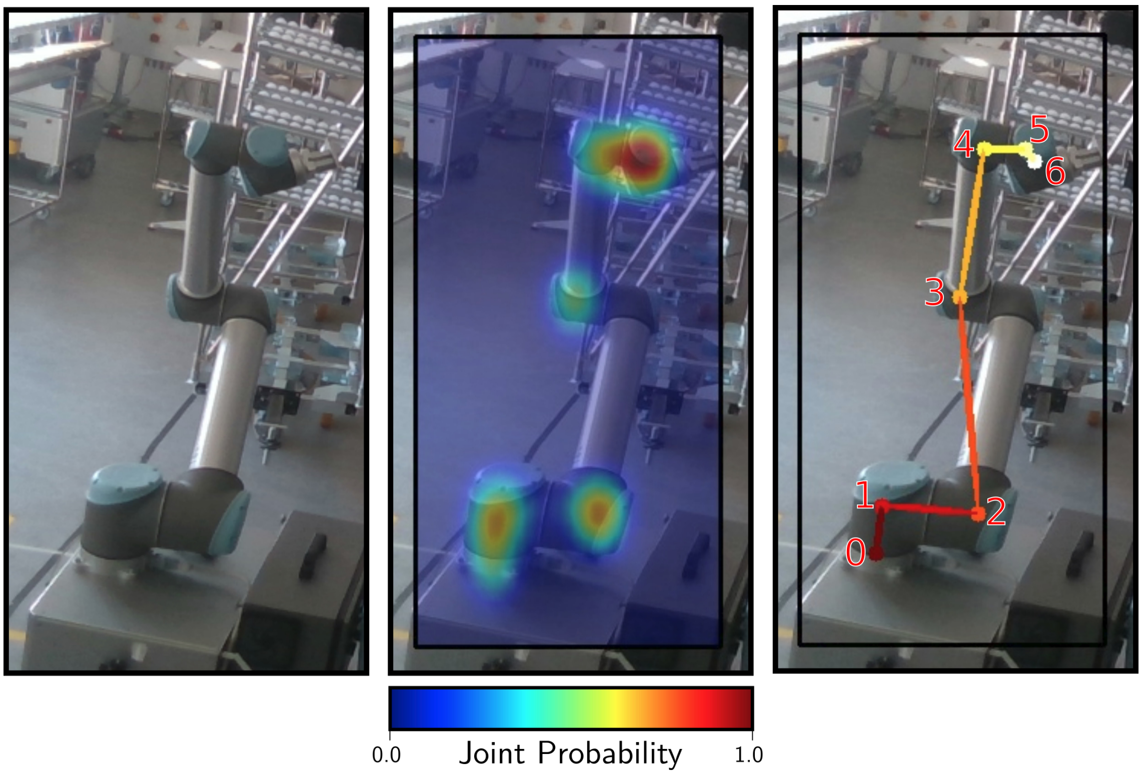


Figure 3.1: The main idea of the *RoPose* system. The intermediate step (middle) extracts individual dense probability heatmaps for each robot joint. The local peaks of these probabilities can then be used to determine the pixel coordinates for each joint position on the image plane.

The here presented idea relies on a CNN structure to predict a dense probability map (see the intermediate step in Figure 3.1) for each joint which is then used to extract the 2D joint positions on the image plane according to the particular highest probability. This procedure can be seen as common practice in various human pose estimation systems [153, 154, 178] and is transferred to robot targets. These resulting joints can then be used to determine the current pose of the individual parts of the robot's kinematic chain. This information can than be fused with the possibly available pose information of relative objects or even humans within the workspace of a robot (cf. Section 4.1).

3.3.2 Data Generation

"Data is a precious thing and will last longer than the systems themselves."

Tim Berners-Lee

Annotated data is crucial for data-driven algorithms. Especially for systems that base their training on full supervision like *RoPose* (see Section 3.3.3), a significant amount of data is needed[1]. However weak- or even self-supervised designs, where the training process or parts of it does not relate to fully annotated data, also need to be objectively evaluated before such systems can be deployed to a real-world application.

At the beginning of this project[2] no publicly available dataset containing the information needed for *RoPose* could be found. Therefore new labeled datasets had to be generated to realize *RoPose* and also for the public community. Besides the reason that it would not be the smartest move to hand-label data for a robotic system that knows exactly all 3D joint poses at any time far more accurate than a human could ever label these positions, other reasons regarding human and financial resources as well as the fact that hand-labeling data is a boring, monotonous and an error-prone job that should be avoided at all costs, it has been decided not to just generate labeled datasets but to design a tool that helps all users to generate their own data. Such a tool would also help to cover new unknown models and very special visual appearances of robot arms caused e.g. by cables, hoses, valves, special gripping systems. The possibility to generate domain specific data also allows possible users to fine-tune the *RoPose* system for their special needs and applications.

To develop an as abstract as possible framework, the ROS-based architecture shown in Figure 3.2 was defined for the data generation pipeline. By using ROS mainly as an abstraction layer and relying on the offered interfaces like the internal transformation system [182], it was possible to use the same pipeline for simulated and real datasets.

[1]There is no generally true answer to the question *"How much data do I need to train my deep neural network?"*. The amount of data heavily depends on the problem that the defined architecture should solve (e.g. number of classes for classification tasks), the targeted domain, the employed data normalization techniques, and many other things. Because this discussion is way out of the scope of this thesis it is kindly referred to other publications treating that topic [179, 180, 181].

[2]And to the best of the authors knowledge, this is still true for the date of the submission of this thesis.

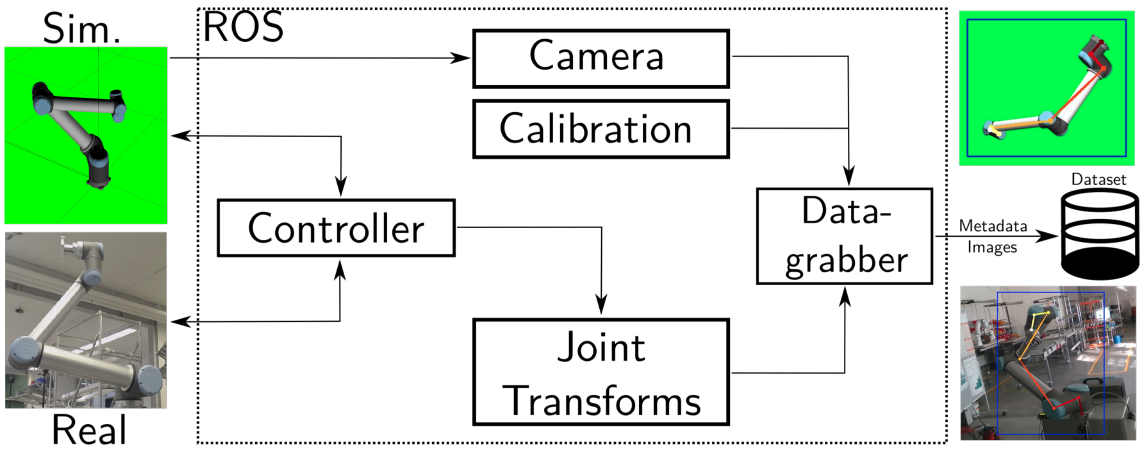


Figure 3.2: Overview of the ROS-based data generation pipeline. The left side shows both, the simulated and real, possible robot targets. Both robot-systems using the exact same controller and rely on real or simulated camera data. The joint transformations are published by the controller and can be stored with the proposed data-grabber system to generate simulated and real-life datasets.

The *datagrabber* monitors the current sensor topic and available joint transformations which are published by the simulated or by the real controller of the robot system. With the available camera calibration information and its extrinsic pose which is also published as a transformation, the particular joint poses can be transformed in 6 DoF either to a global world or to a local camera space coordinates. Together with the projected 2D pixel coordinates (cf. Section 2.4.2 and 2.4.2.1) the dataset is extended frame by frame and continuously stored in the filesystem as described in Section 3.3.2.3 below.

While recording datasets, it is mandatory for the robot system, simulated or real, to change the physical joint positions of the robot constantly. The dataset generator then perpetually grabs the sensor and pose data. To realize this functionality, the controlling paradigms were not re-implemented but the available options from the controller framework *MoveIt!* [11] were exploited to allow for a low level controlling of the robot. The *MoveIt!* based controller can be used on both domains and, thanks again to the ROS abstraction, it can be used for simulated and real robots without the need of any adaption.

The provided data generation pipeline also offers multi-camera support to reduce the consumed time to produce real datasets. This allows for the recording of multiple camera viewports at the same time.

Though the developed data-generator excessively uses ROS-elements, the resulting datasets are completely ROS-independent (cf. Section 3.3.2.3) and can so be used from other frameworks as well. Besides the ROS environment [80] where the systems presented in this thesis were finally targeted, the Visual Servoing Platform (ViSP) [183] also offers a possible platform. ROS was chosen mainly because of its large, growing userbase and also the large number of manipulators that are natively supported by the *ROS-industrial* platform [184, 185].

The *RoPose Datagrabber* [ii], the ROS-independent *RoPose Dataset Tools* [iii], as well as the *Kollrobot Controller* [vii], which can be used to trigger random positions of the robot-arm, were made publicly available.

3.3.2.1 Synthetic Dataset Generation

The synthetic dataset generation is based on the simulation suite *GAZEBO* [186], which is directly integrated into ROS. As the research team had only one *UR10* from *Universal Robots™* [187] available for real data acquisition, the very same robot was used within the simulation domain. Luckily, the manufacturer offered a native ROS-package with ready to use robot models [188] but the proposed system is designed to work with all robot systems that have a ROS driver and *GAZEBO* interface available. Figure 3.3 shows an example scene from the simulation, whereby the proposed abstraction is based on the available transformations as well as final examples of the generated images for the dataset.

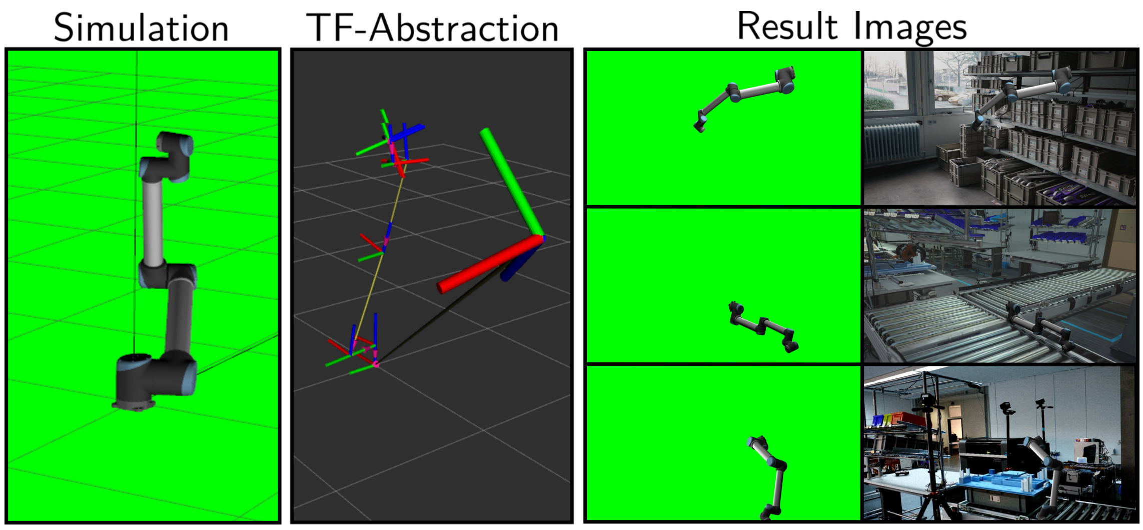


Figure 3.3: Overview of the simulated data generation. The basic simulation (left) is based on *GAZEBO* [186] with a completely green background for chroma key compositing purposes. The labels for the joint positions are abstracted by the ROS transformation system (TF-abstraction) while the resulting images are made available as ROS-independent images. Examples for exchanged green backgrounds with real examples can be seen on the right.

At an earlier point, it was decided to put no more effort into the creation of suitable backgrounds inside the simulation environment as *GAZEBO* is (or at least was back in 2018) limited when it comes to rendering and scene creation. The then introduced simulated green screen background (also shown in Figure 3.3) passes the workload of the background generation to the final user of the simulated datasets. The open-sourced python package *RoPose Greenscreener* [iv] is capable of extracting the background employing chroma key compositing methods, can exchange the background with available images and also can be used to place other robots as additional clutter in other images.

One of the biggest advantages of simulating vision-based data is that the pose of the simulated vision sensor relative to the target poses is always known. This information is essential for a detailed dataset where the labels should contain 6-DoF poses and also the corresponding 2D projection on the respective camera frame. This makes it easy to dynamically change the viewport of the camera which is an important parameter to generate data with sufficient variations and to create an ideal viewport-independent model. Because *GAZEBO* does not offer this functionality, the following procedure to calculate a random camera position for each frame was integrated. Internally, all linear algebra operations were based on the *Eigen* template library [189].

- **Defining a new random 3D position vector ($\mathbf{p}_{cam}$) of the camera.**
  For each frame, a random 3D position vector $\mathbf{p}_{cam}$ for the simulated camera is generated. This position is calculated by defining a virtual spherical coordinate system with it's origin in the robot's base and by choosing random values for the radius r, the polar angle φ as well as the azimuthal angle θ. The random coordinate can then easily be converted into the Cartesian based system used by the simulation (cf. Section 2.2). The simulation mode of the *RoPose-Datagrabber* offers parameters to limit r, θ and φ to reasonable values to avoid unrealistic viewports[3]. Within the set limits, the random values are generated based on a standard uniform distribution.

- **Calculating the orientation to focus the view on a specific 3D point.**
  For the complete 6DoF pose of the sensor also a rotation is needed which ideally keeps the robot inside the viewport of the camera. As the position of the camera

[3]In practice the values were limited as follows: $2.0m < r < 8.0m$, $0.3 < \theta < \pi$ and $0.0 < \varphi < 2\pi$.

($\mathbf{p}_{cam}$) and the base position of the robot ($\mathbf{p}_{base}$) are known, the *look-at*-rotation can be calculated as follows.

$$\mathbf{r}_x^{'} = \mathbf{p}_{base} - \mathbf{p}_{cam} \tag{3.1}$$

$$\mathbf{r}_x = \frac{1}{||\mathbf{r}_x^{'}||}\mathbf{r}_x^{'} \tag{3.2}$$

$$\mathbf{r}_y^{'} = \mathbf{r}_x \times \mathbf{t}_{rand}, \text{ with } 0 \leq t_x, t_y, t_z \leq 1.0, \mathbf{r}_y \neq \mathbf{r}_x \tag{3.3}$$

$$\mathbf{r}_y = \frac{1}{||\mathbf{r}_y^{'}||}\mathbf{r}_y^{'} \tag{3.4}$$

$$\mathbf{r}_z^{'} = \mathbf{r}_x \times \mathbf{r}_y, \text{ with } \mathbf{r}_z \neq \mathbf{r}_x, \mathbf{r}_y \tag{3.5}$$

$$\mathbf{r}_z = \frac{1}{||\mathbf{r}_z^{'}||}\mathbf{r}_z^{'} \tag{3.6}$$

$\mathbf{r}_x$ describes the normalized distance vector between the camera's position and the robot's base while $\mathbf{r}_y$ is defined by the normalized cross product of $\mathbf{r}_x$ with a random normalized vector $\mathbf{t}_{rand}$ and can be seen as the random UP-vector which basically defines a random rotation around the optical axis of the camera. Finally $\mathbf{r}_z$ will result in the vector that is orthogonal to the virtual plane described by $\mathbf{r}_x$ and $\mathbf{r}_y$.

Finally the new *look-at*-rotation matrix $\boldsymbol{R}_{lookAt}$ can be constructed with the coefficients of the normalized rotation base vectors $\mathbf{r}_x$, $\mathbf{r}_y$ and $\mathbf{r}_z$

$$\boldsymbol{R}_{lookAt} = \begin{pmatrix} r_{x0} & r_{y0} & r_{z0} \\ r_{x1} & r_{y1} & r_{z1} \\ r_{x2} & r_{y2} & r_{z2} \end{pmatrix} \in SO(3) \tag{3.7}$$

To avoid that the robot's base always being located directly in the middle of the defined viewport, the system offers the possibility to randomly shift the *look at point*. This is simply realized by adding a random offset δ to each vector coefficient of $\mathbf{p}_{base}$[4].

Notably, this procedure assumes that the robot's base is located at the world system's origin. To realize this procedure in different coordinate systems, additional coordinate transformations have to be performed (cf. Section 2.1).

The authors first topic related publication [I] is targeted completely in a simulated domain and may be worth a read if further information is needed.

[4]In practice the random offsets were limited for each coefficient to $-2.0m < \delta < 2.0m$.

3.3.2.2 Real Dataset Generation

When working with real datasets it is clear that the only missing element in the data production pipeline visualized in Figure 3.2 is the data related to the sensor hardware - mainly the extrinsic pose[5] relative to the robot's intrinsic coordinate system. One of the big benefits of *RoPose* is the use of the extracted pose information to perform an extrinsic calibration of the camera (cf. Section 3.4). However, as there is no way to exploit this advantage before enough data to train the estimator itself is available, this *chicken and egg* problem has to be solved and alternative calibration procedures have to be considered and integrated.

In robotics, or especially for industrial robot arms, the problem of finding the correct extrinsic relations between the camera and the robot system, is called *hand-eye*-calibration. This term is used for both possible set-ups where the camera is mounted somewhere on the robot itself (*eye in hand*) or when the sensor is not connected to the kinematic chain of the actuator at all and serves as a supervising sensor just as in our main use case. There are several known ways to estimate the pose of a single 2D-vision sensor relative to specific coordinate frames that have been published in the last decades and most of them are based on known calibration targets like chessboards [190, 191, 192].

As it is one of the main ideas behind *RoPose* to avoid the need for other hardware equipment to use the provided functionalities a pure visual approach needs to be introduced. At this point the beforehand doomed manual labels finally become helpful. The idea is simple: Provide a minimal set of hand-labeled images and perform the extrinsic calibration for a specific perspective based on the now available 2D and 3D correspondences. As further explained in Section 3.4, the given task can now be solved by modeling a PnP-Problem (cf. Section 2.4.3). This allows for an extraction of the camera transformations, on which the further dataset creation is based on, with the help of an additional software tool that offers a minimal user-interface. An example of the manual labeling process can be seen in Figure 3.4[6].

[5]It is assumed that the intrinsic camera parameters are known

[6]The official supplementary video created for *ICRA2019* gives a good example of the calibration process and can be found at *IEEEExplore*: https://ieeexplore.ieee.org/document/8793900/media

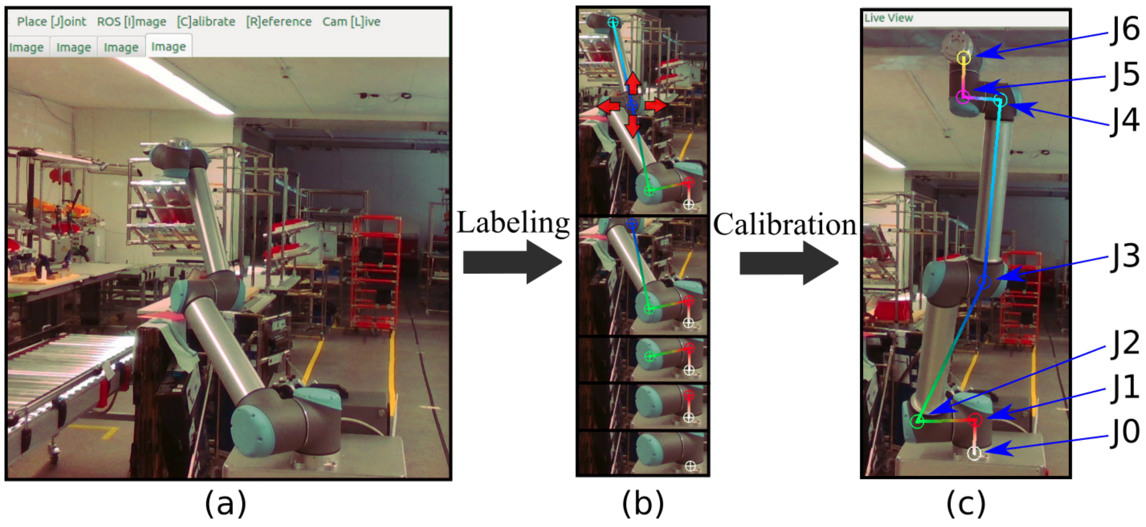


Figure 3.4: Extrinsic calibration process by hand-labeling the robot poses. (a) Collected raw input image. (b) Indicates the labeling process for every single robot joint. (c) Reprojected live pose after a calibration was performed. Source: Figure 2 in [II] © 2019 IEEE.

The novel tool, that has been introduced in the second publication [II], offers the possibility to capture multiple images together with the corresponding 3D joint positions and label the visible joint positions afterwards. After performing the labeling, the calibration algorithm that internally uses the approach Lepetit *et al.* proposed [103] and is available in *OpenCV* [193] is used to estimate the extrinsic pose of the camera system. The pose information is automatically published as a ROS-transformation and can be directly exploited by the dataset generator. Following this calibration process, a live view of the camera shows the reprojected 3D joints of the robot (Figure 3.4 (c)) that can be used to subjectively check if the estimated camera pose is good enough. To obtain good calibration results, it is recommended to use as diverse robot poses as possible to reduce the perspective ambiguities. Objective evaluations of the hand-labeled calibration procedure have been published in [II].

Alternatively to the manual labeling, every possible method to obtain the correspondences can be used as long as the transformation of the camera is published in the supposed way. To avoid excessive manual labeling when recording multiple datasets from many different camera perspectives, an *AprilTag* [9, 10] based calibration procedure, where a single tag has to be placed on a known position on the robot arm (usually directly at the robot's flange), was also realized and integrated into [vii]. This approach was chosen because *AprilTags* [194, 195, 196] and *AR-Marker* [197] were used in various application as extrinsic references to perform many different calibration tasks. When a tag can be detected within an image, the calibration system stores the 2D origin of the tag ($\mathbf{p}_{tag}$) and the corresponding 3D position $\mathbf{p}_{rob}$ in the robot coordinate system. To care for a better spatial distribution of the correspondences, an additional distance constraint was integrated (see Eq. 3.8) by checking the 3D distance d between each new candidate $\mathbf{p}_{new}$ and each of the N stored correspondences (C) in the robot frame. These again can be calculated based on the Euclidean distance in $\mathbb{R}^3$.

$$\begin{gathered} dist(\mathbf{p}_{new}, C) = \|\mathbf{p}_{rob} - \mathbf{p}_{new}\|_2 \ , \ \mathbf{p}_{rob} \in C \\ d_{min} = min(dist(\mathbf{p}_{new}, C)) \end{gathered} \tag{3.8}$$

$$\text{with } \|\ldots\|_2 = \text{Euclidean Norm}$$

After collecting various correspondences from N frames[7], the standard calibration procedure that was already used as the last step for the manual labeling procedure is performed to estimate the needed camera pose.

Mainly various versions of *Intel® RealSense™* sensors [198] were used to generate real datasets. Figure 3.5 shows data samples from the published collection. This sensor offers many different frame types just as RGB, infrared, and depth, in a handy design and provides a fully integrated ROS-wrapper[199] which makes it the perfect sensor candidate for the here presented research. However, the system has also been tested with Microsoft Kinect sensors [200] and simple standard webcams. Every sensor that can be extrinsically calibrated in any way and is able to publish the needed information to the relevant ROS-topics for a specific frame type, should be fine.

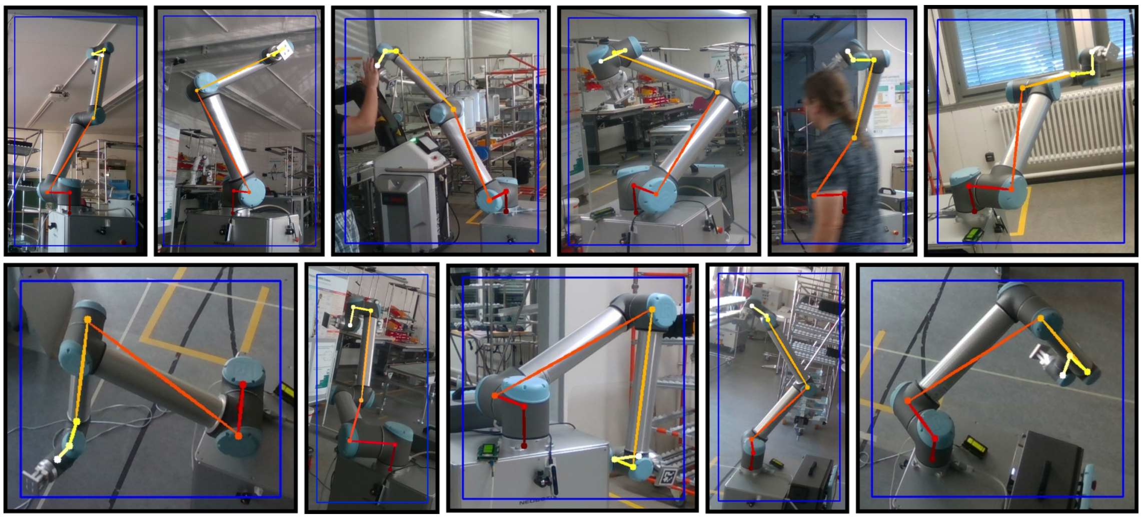


Figure 3.5: RGB examples of generated real datasets from various camera viewports.

The second *RoPose* related publication [II] covers the real data generation and the adaption of the pose estimation to real datasets in detail.

[7] In practice, usually $N = 50$ correspondences with a minimum distance of $d = 0.05m$ were collected.

3.3.2.3 Datasets

Due to the identical generation pipeline the simulated and real datasets have the same structure and organization and can be used and integrated into new projects using the same code base that have been published in [iii]. Each *RoPose* dataset contains the following data if the used hardware is able to provide the specific frame data.

- **Sensor data**
  For sensor data the system supports normal RGB- and infrared images and also depth measurements. The actual raw data are kept in the particular typical data format and will be stored in a separate folder for each configured sensor frame.

- **2D and 3D joint labels in different coordinate frames**
  For every frame, independent of the particular type, the recorder will create a strictly numbered *JSON*-file containing all the local labels ($\mathbf{p_{3D}} \in SE(3)$ and $\mathbf{p_{2D}} \in SE(2)$) in the sensors specific coordinate frames as well as additional information like a time-stamp and if the recorded robot pose was somehow redundant. The redundancy parameter was implemented because it could take some time to calculate a new valid path to a random position. Within this timespan, the robot's pose would not change. The labels in global world coordinates (in a recording scenario this is the robot frame) will also be stored in a separate file.

- **Additional Metadata**
  Metadata to define fundamental parameters of the dataset was also added. The *metadata.json* file contains information of the robot type, if the particular dataset is synthetic or not, and if the dataset was recorded with a green screen which could theoretically also be used with real robots.

After some development iterations, the open-sourced data grabber mainly targeted for robot arms has evolved into a fully configurable dataset generator where a final user can choose the data-sources that needs to be recorded. For more information, a detailed README is provided in the related repositories [ii, iii]. All datasets generated within this project where made publicly available online[8].

[8]https://www.thomas-gulde.de/ropose/downloads

3.3.3 Keypoint detection

3.3.3.1 Related Work

The *RoPose* estimation can be seen as an abstract keypoint or joint detector of an articulated object or target and is thus related to any other keypoint estimator. Therefore, the most prominent use case that has received the most attention in the last years from many research teams around the globe is the pose estimation of humans [201, 202, 203, 204]. The range of use for such tools varies from a single instance focus to even large scale crowds [205, 206].

In the age of deep learning and CNNs, traditional computer vision methods were pushed in the background, especially when the task is somehow related to pattern analysis, recognition, or classification. Amongst other indicators, many benchmarks [207, 208, 209] from different fields of research[9] confirm that deep learning or CNN based approaches clearly outperform their traditional opponents, at least in reference to the final accuracy. Despite that, Walsh *et al.* recently outlined that not all known classical methods have become obsolete [212]. Hybrid approaches where conservative and modern deep learning methods are based on each other can also be seen as a solid base for future applications.

The first *RoPose* models (cf. [I, II]) where based on the work of Wei *et al.* [153] and Cao *et al.* [154], where a recurrent CNN design was used to extract the human joint probabilities and the so-called *affinity maps* to reconstruct the poses from multiple human instances within an image. *Stacked-hourglass* architectures (cf. Section 2.6.2.2) networks like [138, 213, 214] have also successfully competed on special pose estimation challenges [54, 207] and extract the probabilities of body joints with a sequence of convolutional- and deconvolutional layers to reduce and extend the trainable feature kernels. Besides all these more or less complicated network structures, Xiao *et al.* presented their *simple-baseline* structure [178]. The architecture adds a few trainable deconvolutional layers after the base feature extraction to extract the probability heatmaps. Because of this simplicity and the fact that the introduced method reached close to SOTA performance, the current *RoPose* implementation is based on this architecture (cf. Sections 3.3.3.2 and 4.1).

Most of the presented related works are fully supervised methods. However, in the decade of GANs, some publications that based the keypoint detection on two internally competing networks should also be mentioned [215, 216, 217, 218].

[9]Like already mentioned for object detections (cf. Section 1.3), at the day of submission of this thesis, the leaderboards of common challenges [210, 208, 211] where lead by deep learning, respectively CNN based approaches.

3.3.3.2 RoPose and Heatmap Regression

After solving the problem of missing datasets with the presented dataset generators (cf. Section 3.3.2), the final pose estimation was developed. As *RoPose* was under continuous development since the first simulation-based approach was published [I] and adapted the same estimation system to real data in [II], this Section will focus on the final proposed system which is now based on the new simple baseline structure introduced by Xiao *et al.* [178]. Besides an additional prediction of the ROI of different targets (cf. Section 3.3.5), the changes made to adopt the new CNN structure, almost completely affect the final heatmap regression. The core of the most parts of the processing pipeline, like the main data handling (cf. Section 3.3.3.3), the training procedure and loss function (cf. Section 3.3.3.4), as well as the post-processing (cf. Section 3.3.4) could be preserved. The flow chart in Figure 3.6 visualizes the needed steps to estimate the probabilities of a target's joints.

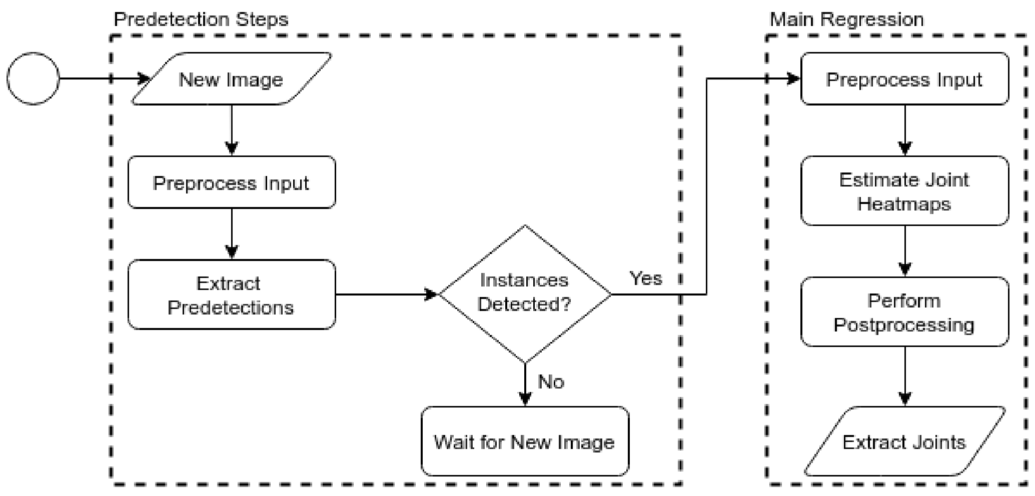


Figure 3.6: Flow chart of the proposed *RoPose* detection pipeline. The additional steps have to be performed to extract ROIs of possible target instances while the joint estimation itself is performed for each detected instance given by the pre-detection.

The proposed network structure by Xiao *et al.* [178] first relates on a feature extracting backbone. As the older *RoPose* regression system used the *VGG16* structure [130] the current system builds upon a *ResNet50* feature extractor [137]. As the original authors tested their system with various backbone versions, *RoPose* follows the suggested method (designated as method (*a*) in [178]) which seems a good compromise between accuracy and speed for the proposed system, according to the provided evaluation results in the original paper. The top layers of the *ResNet50* was dropped and the output of its *C5* layer (cf. Table 1 in [137], named *conv5_x*) form the feature input for the following three de-convolutional layers with a kernel size of 4×4, followed by a 1×1

dense convolutional output-layer to allow for the needed regression to the heatmaps. For all de-convolutional layers, standard *ReLU* [115] was used as activation functions followed by a 2D batch normalization [128]. To generate the heatmaps, the final output convolutions' activation function is simply linear. The final regression architecture is shown in the following Figure 3.7, more details about the structure can be found in the original work [178].

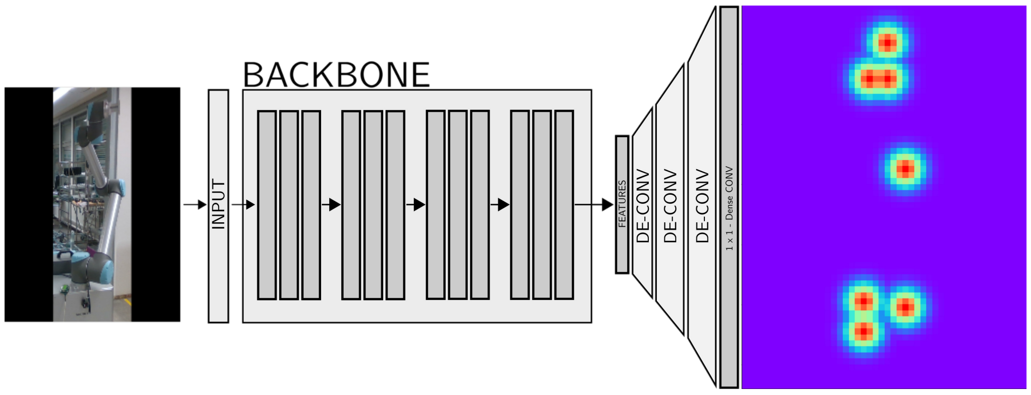


Figure 3.7: Final *RoPose* regression architecture based on the simple heatmap regression pipeline presented by Xiao *et al.* in [178]. After the ROIs of possible targets are extracted, cropped, and preprocessed from the raw input image, they will be fed into the *ResNet50*-backbone [137] to extract the base features. The added three de-convolutional layers are followed by a 1×1 convolutional output-layer, which finally performs the regression, will produce the ready to use dense probability maps.

As suggested in method (*a*) in the original work [178] the shown network structure is designed to take preprocessed (cf. Section 3.3.3.3) three-channel RGB input of the size 256×192 and produces eight heatmaps of the size 64×48 which can be used to extract the final joint positions (cf. Section 3.3.4).

At the very end, this setup produces an independent dense probability map for each joint (cf. Figure 3.8) and extracted ROI which is used as a base for the post-processing steps to extract the final 2D pixel position for each joint.

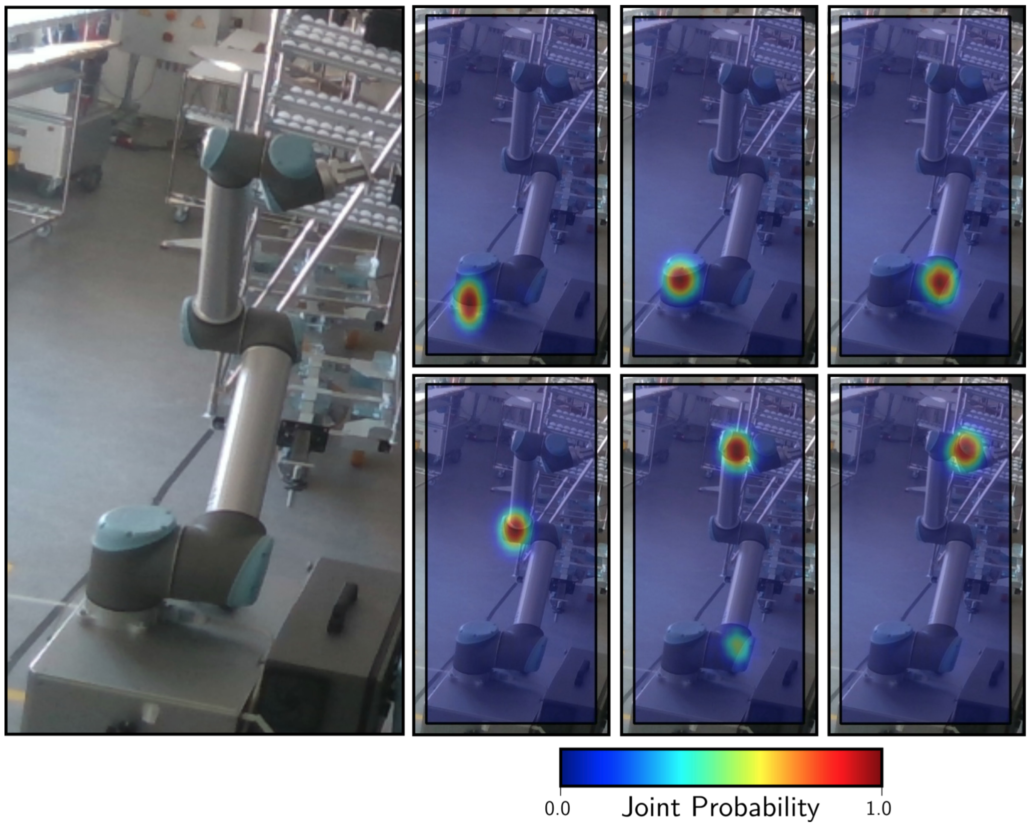


Figure 3.8: Examples of predicted dense probability heatmaps for each of the robot's joints.

The final, meanwhile *PyTorch*-based [126] implementation covering the here presented system were made publicly available in [i].

3.3.3.3 Data Handling, Preprocessing and Ground Truth

No matter what kind of NN is involved, the correct handling of the available data sources is a crucial part of the system. A deep learning-based estimation system like *RoPose* can only work if the data representation for the network's input and output is clear and followed strictly, no matter if the network is trained or just used for prediction.

As introduced in Section 3.3.3.2, the final regression network expects only the extracted ROI of the pre-detected robot-arm (Figure 3.10(a)) in form of an RGB image. Generally, an image (img) is represented by three channels (c) containing the color values of each pixel (p_{uv}), usually in form of 8 bit unsigned integer values ranging between 0 and 255 ($int_{min} - int_{max}$). This range and data format is not perfect for NNs that extensively use floating-point operations. However, the first simple preprocessing step is to cast the input's channel values into floating points and normalize them to a well-arranged value range. This normalization can easily be performed by dividing each pixel in each channel by int_{max} producing a color value range of $0.0 - 1.0$ suitable for the network use[10].

$$p'_{uv} = \frac{p^c_{uv}}{int_{max}} \tag{3.9}$$

In order to stick to the targeted network's input size of 256×192, the image has to be down- or upsampled, depending on the relative size of the bounding box. Different to Xiao *et al.* [178] where the bounding box itself was modified to fit with the network's aspect ratio and so also automatically conserves the aspect ratio of the original image, *RoPose* employs a zero value padding mechanism as shown in Figure 3.10(b), to keep these ratio dimensions. The padding mechanism places the cropped image in the center of an input matrix with the aspect ratio of the network's planned input size and fills all not occupied pixels with a constant value ($c_{pad} = 0.0$) as pictured in Figure 3.9. To quantify the added padding, a 1×4 vector $\mathbf{C}_{pad}$ is used to describe the amount of pixels used for the padding on each side (left, right, top, bottom) of the image.

$$\mathbf{C}_{pad} = \begin{pmatrix} C_{pad,left} \\ C_{pad,rigth} \\ C_{pad,top} \\ C_{pad,bottom} \end{pmatrix} \tag{3.10}$$

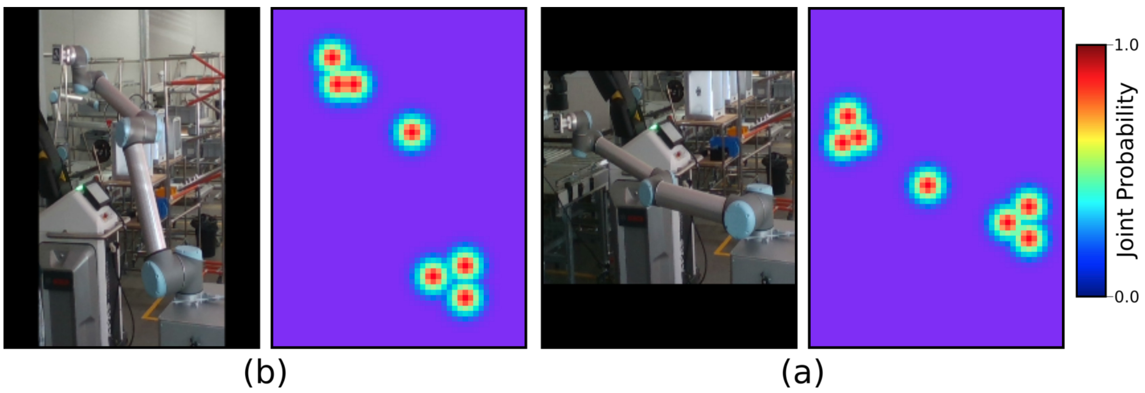


Figure 3.9: Impact of the padding mechanism on the input and GT data. (a) padding along the x-dimension, (b) padding along the y-dimension.

[10]This range is not a hard specification. It is also common practice to range an input between -1.0 and 1.0 or -0.5 and 0.5.

After the padding, the resizing factor ($\mathbf{f}_{resize}$), used to resize each ROI to the target size, can be calculated with the known dimensions of the padded ROI ($\mathbf{S}_{roi}$) and the static input size of the network ($\mathbf{S}_{inp}$).

$$\mathbf{f}_{resize} = \begin{pmatrix} f_x \\ f_y \end{pmatrix} = \begin{pmatrix} \frac{S_{roi,x}}{S_{inp,x}} \\ \frac{S_{roi,y}}{S_{inp,y}} \end{pmatrix} \tag{3.11}$$

To transfer these preprocessing steps to a particular GT, all factors calculated for the input image have to be applied to the original 2D joint information based on the image plane coordinates ($\mathbf{p}_n$) of the original image stored in each dataset, relative to the upper-left coordinate of the target's bounding box ($\mathbf{p}_{bb}$).

$$\mathbf{p}'_n = \left(\mathbf{p}_n - \mathbf{p}_{bb} + \begin{pmatrix} C_{pad,left} \\ C_{pad,top} \end{pmatrix}\right)\begin{pmatrix} \frac{1}{f_x} \\ \frac{1}{f_y} \end{pmatrix} \tag{3.12}$$

The projected and preprocessed joint information from the available datasets and the origin coordinates of the bounding box are used to generate the final regression target by placing 2D Gaussian blurred peaks at the absolute pixel coordinates of each joint. As such a GT preparation is used by various keypoint estimation systems (e.g. [153, 154, 219, 138]) this procedure can be seen as a common practice in this research field. While Figure 3.10(c) shows an example of the combined regression targets based on the input resolution, Figure 3.10(d) demonstrates the final combined heatmaps based on the output size (64×48) of the network and represents the direct supervised training input of the network.

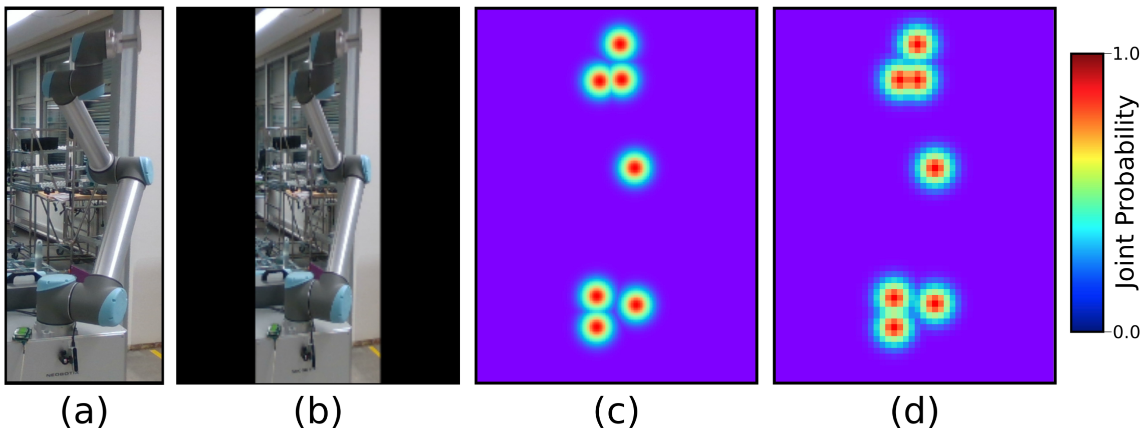


Figure 3.10: Data representation for the input and GT used to train *RoPose*. (a) Raw input created by cropping the image with the bounding box of the target. (b) Resized and padded network input (size: 256×192). (c) Combined generated GT in input resolution. (d) Combined resized GT to fit the network's output resolution (size: 64×48). Note: This figure shows the combined heatmaps for the sake of clarity, the actual GT data passed to the network consist of multiple heatmaps, containing one peak per heatmap and an additional segmentation mask for the background.

When looking at the required preprocessing, it does not matter if the architecture of the first *RoPose* model [I, II] is considered, or the current implementation. Both versions build upon a well known deep feature extraction backbone which demands special pre-processing to benefit from the pre-trained network weights and allow for a transfer of these features to our specific problem. Luckily, both used backbones [137, 130] employ the identical preprocessing. The preprocessing step consists of an additional normalization for every sample. This normalization focuses on the different color channels and is based on the extracted color mean and standard deviations of a specific dataset (cf. *ImageNet*[55]) and is thus an essential data preparation step that has to be integrated in the system.

3.3.3.4 Standard Training Routine

The GT that was used to teach the employed CNN the final regression to the heatmaps was already detailed and can be created for each example as described in Section 3.3.3.3. The bounding box used to determine the ROI around each example was directly taken from the annotations and is therefore not based on other algorithms. To simulate a more general dataset, traditional image data augmentation methods based on affine transformations (see Section 2.1.4), were used. Figure 3.11 shows the influence of the various transformations on the input and GT data.

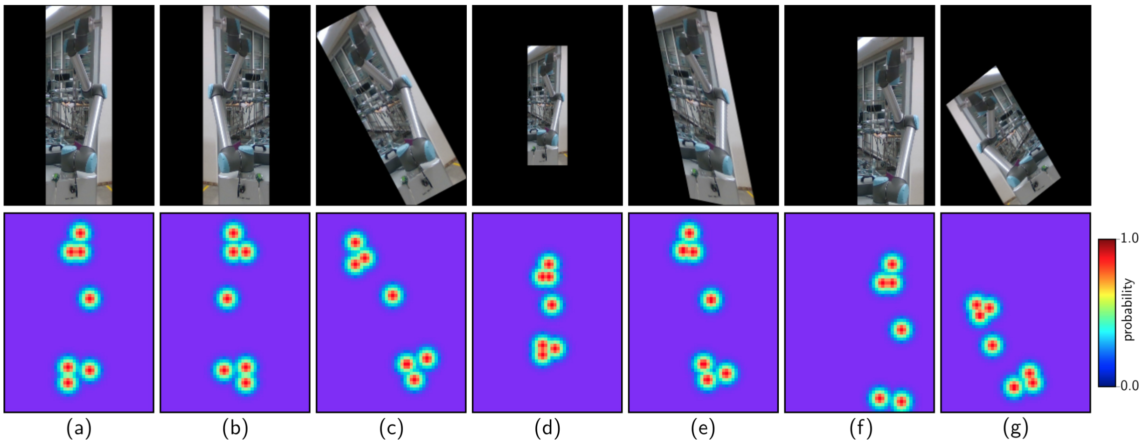


Figure 3.11: Visualization of the augmentation impact for the input (upper row, size: 256×192) and the related GT (size: 64×48). (a) non augmented reference input and GT, (b) flipped, (c) rotated ($\alpha = -35°$), (d) scaled ($s_{x,y} = 0.7$), (e) sheared ($sh_{x,y} = 0.2$), (f) displaced ($x, y = 40\ pix$), (g) combined affine augmentation.

Besides the employed affine augmentation strategies, additional augmentation techniques [220], like adversarial data augmentation [221, 222, 223] could be used and may be integrated within the training procedure to increase the estimation performance [224] in the future.

The standard loss-function (f_n) that will be minimized during the training is defined by Eq. 3.13 and can also be seen as common practice because it is used in various related applications [225, 138] as well as in the original work [178] of the adopted regression network. f_n is used to calculate the summed quadratic difference between each probability GT p^t and prediction p^p of the target heatmaps (with $v_{max} = 64, u_{max} = 48$ corresponding to the output size of the network). This basically transfers the standard MSE as shown in Eq. 2.94 to a multidimensional tensor.

$$f_n = \frac{1}{D}\sum_{h=1}^{H}\sum_{u=1}^{u_{max}}\sum_{v=1}^{v_{max}}(p^t_{uv} - p^p_{uv})^2 \tag{3.13}$$

with H = number of joint probability heatmaps and
D = overall dimension of the tensor

In order to optimize the deployed model, standard SGD [141] (cf. Section 2.6.3) was used. For the SGD based training, the learning rate (lr) for each epoch $(t_{epoch}$, starting with $1)$ is defined by decaying the start learning rate lr_0 by an exponential factor calculated as shown in Eq. 3.14 where k denotes a decay.

$$lr = lr_0 * e^{-kt_{epoch}} \tag{3.14}$$

The final *RoPose* model presented in this thesis was trained for 400 epochs with a batch-size of 80 while r^0_{learn} was set to 0.001 and the decay factor k to 0.001. The model was trained with enabled augmentation that involved scaling $(\pm 10.0\%)$, flipping, rotation $(\pm 180\,\text{deg})$ and translation$(\pm 10.0\%)$. The implementation of the augmentation pipeline itself is based on the *imgaug* Python-library [226].

3.3.4 Post-Processing

After the network performed the regression and estimated the probability for each joint in separate heatmaps, additional post-processing steps need to be performed to extract the final 2D position. The main step is the non-max suppression to extract the peak probabilities, the maximum value, within each heatmap. This action will extract the coordinate indices ($\mathbf{I}^n_{x,y}$) of the maximum value within each estimated heatmap (H_n).

At this point, the system's design offers some additional possibilities to improve the final estimation accuracy. Instead of taking the indices of the maximum raw output as the basis for the pixel estimation, the output heatmaps can be upsampled to fit e.g. the original input size of the estimation network. This leads to a subpixel accuracy compared to the raw output heatmaps and should increase the detected peak's accuracy.
To upsample 2D data various established methods like 2D nearest neighbor-, bilinear, or bicubic interpolation exist. As the nearest neighbor approach would just copy the potential highest peak value of a joint to a dense area, it was not considered as a suitable technique and bilinear and bicubic techniques were used. Examples of the various upsampling results are shown in Figure 3.12.

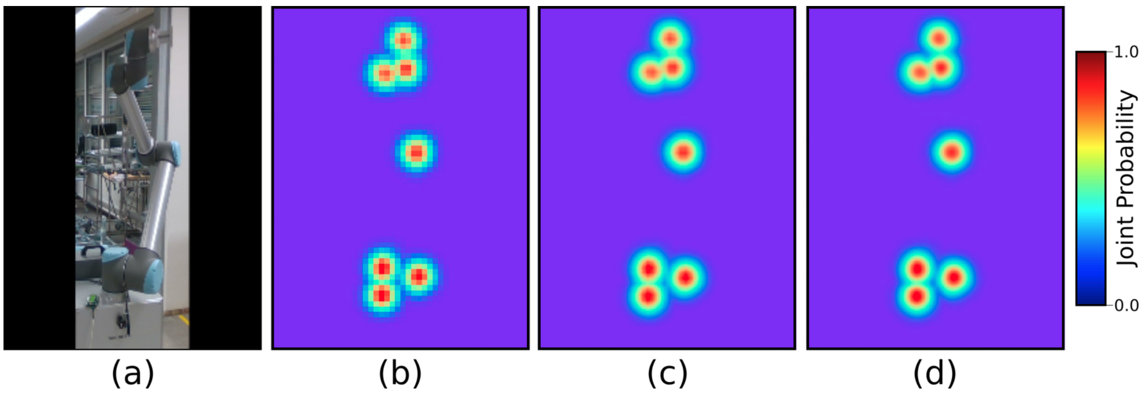


Figure 3.12: Accuracy impact of upsamling the raw output with different allgorithms. (a) Raw input fed into the network (size: 256×192). (b) Raw output from the network (size: 64×48). (c) Upsampled output (bilinear) (size: 256×192). (d) upsampled output (bicubic) (size: 256×192).

Visually, it seems there is almost no difference between bilinear and bicubic based upsampling. As usual, the naked eye is not the best judge and the final accuracy impact on the joint estimation accuracy was analyzed and discussed within the evaluation in Section 5.5.

However, because such an upsampling operation has to be applied on all output heatmaps, additional computation power is needed which will negatively affect the framerate of the final application. Besides the upsampling itself, an increased heatmap size will increase the size of the pixel probabilities that have to be considered by the non-max suppression that naturally will increase the computation time for this operation as well.

Besides *OpenCV* [193], *SciPy* [117] also offers modules to perform the upsampling on the Central Processing Unit (CPU) and were used within the implementation of the postprocessing. In the final application, the upsampling was directly embedded in the network structure after training the network (cf. Section 6). This enables a Graphics Processing Unit (GPU) based upsampling of the data, leaving the impact to the computation time mainly to the non-max suppression algorithm, compromising between performance and accuracy.

The identified maximum probability position for each of the N joints ($\mathbf{p}_n$) is only valid within the cropped ROI used as input for the network. In order to transfer the extracted joint positions to positions valid for the original input image ($\mathbf{p}'_n$), all the performed preprocessing steps (cf. Section 3.3.3.3) have to be reversed. This is more or less trivial but requires the distinguished storage of all parameters used for the preprocessing, such as the input resize-factor $\mathbf{f}_{resize}$, the absolute applied padding ($\mathbf{c}_{pad}$) for every border of the image, and the position of the upper left corner of the extracted ROI on the image ($\mathbf{p}_{roi}$).

$$\mathbf{p}'_n = \mathbf{p}_{roi} + \mathbf{p}_n \mathbf{f}_{resize} - \begin{pmatrix} \mathbf{c}_{pad,left} \\ \mathbf{c}_{pad,top} \end{pmatrix} \tag{3.15}$$

3.3.5 Bounding Box Estimation

As the bounding box detection can be seen as a pre-detection system to estimate the raw ROI for a robot system of interest, this step of the estimation is completely independent of all other procedures that are performed within the estimation task. There are versatile common prediction systems available. The *R-CNN* published by Girshick *et al.* [227] and also the refurbished *Faster-R-CNN* [228] provide one of the first deep learning-based bounding box detectors used for multiple object detection. As the original *R-CNN* work completely lacks real-time performance because it is based on a huge amount of independent region proposals per image which are also evaluated by a CNN. The *Faster-R-CNN* allows much faster predictions by employing ROI-pooling directly on the feature maps which are extracted from the raw input image.

Another prominent example is the *single shot multibox detector*. The estimator published by Liu *et al.* is generally faster as the known *R-CNN* based systems but shows a slightly lower accuracy [229]. Like many other applications, this model employs a *VGG* model [130] to extract base feature maps and adds additional convolutional layers to classify the trained classes and perform the final regression for the bounding box areas.

In 2016, even before the *single shot multibox detector* were available, Redmon *et al.* published the first version of You Only Look Once (YOLO) [230, 231, 232]. Their system still delivers near SOTA performance and was under heavy development in the last years. YOLO is now available in its official fourth version, is highly adjustable in parameters like input size, and outperforms most of their counterparts when it comes to computational performance [233]. The compromise between close to SOTA accuracy and also very fast computation makes YOLO the perfect candidate for the pre-prediction task and was thus chosen instead of the known alternatives.

In order to efficiently transfer the estimation architecture from [I, II] to the new structure, the publicly available *PyTorch* implementation of *YOLOv3* from *Ultralytics LLC*[11] [234] was wrapped into the *RoPose* project and network structure.

The original YOLO model does not contain a class to predict the positions of industrial robot arms. To enable the needed detection, the network model needs to be retrained and extended by an additional robot class. As described in Section 4.1, the possibility to detect additional classes such as humans will become handy in extended use cases. As in our special case, YOLO only needs to distinguish between humans and robot-arm targets, the network was adjusted to the needed two-class detection and was trained with all images of the 2017 *COCO*-dataset [54] which was extended with the generated *RoPose* data. Some Augmented training examples can be seen in Figure 3.13.

[11] `https://github.com/ultralytics/yolov3`
Commit: 98068efebc699e7a652fb495f3e7a23bf296affd

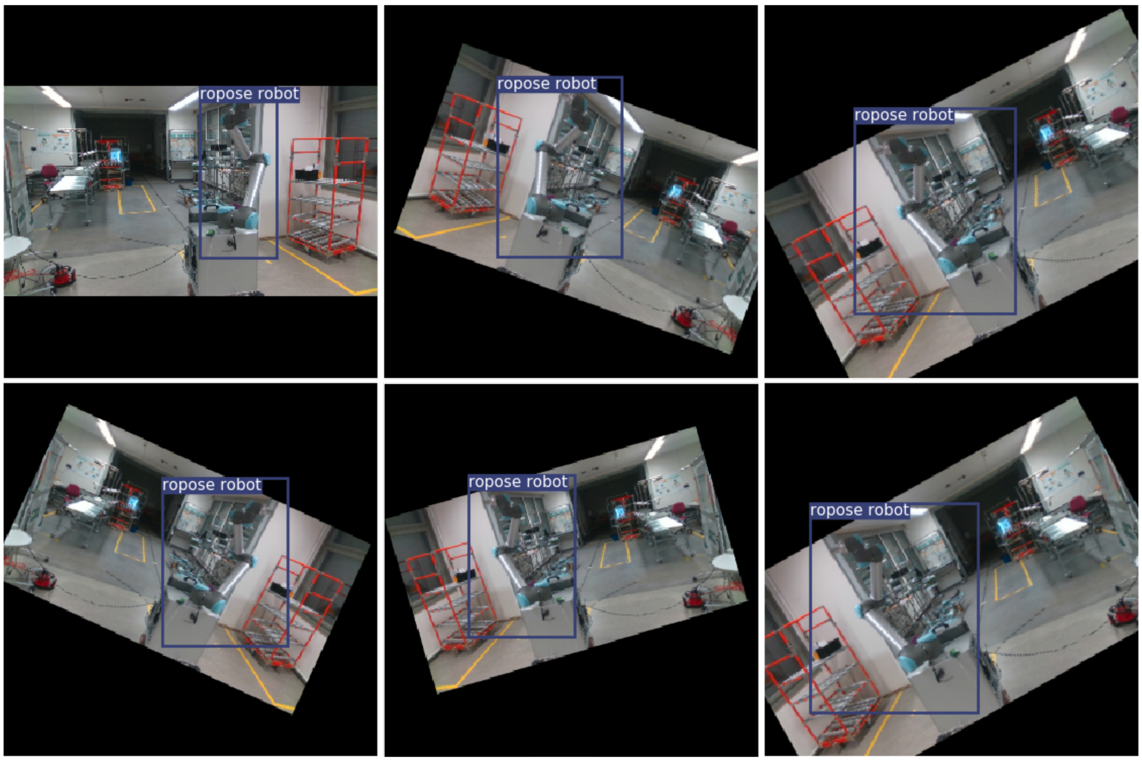


Figure 3.13: Example of the data used to train the bounding box detector. For augmentation also randomly combined affine transformations were used.

As suggested in the original implementation [234], SGD based optimization with a start base learning rate of $r^0_{learn} = 2.324e^{-3}$ and momentum $m = 0.97$ was used for training. Because no problems with the detection of the original *COCO*- human class were noticed, no explicit evaluation was performed and the evaluation coming with the used implementation should be comparable to the resulting model. For the newly added robot-class, the performance evaluation is detailed in the specific evaluation Section 5.3.

3.4 Automatic Extrinsic Camera Calibration

Some of the most important information to know when observing a robot's workspace with a vision sensor is the transformation of the 6DoF pose of the sensor relative to the robot's base coordinate system. Only when this transformation is available, it is possible to transfer positional information from the image plane to the robot workspace. To estimate this pose, the extracted keypoint information of the here proposed *RoPose* system can be fused with the 3D positioning information of each joint, which are accurately known by the intrinsic robot controller[12]. The extrinsic relations, shown in Figure 3.14, were also exploited for the labeling based camera calibration procedure presented in Section 3.3.2.2.

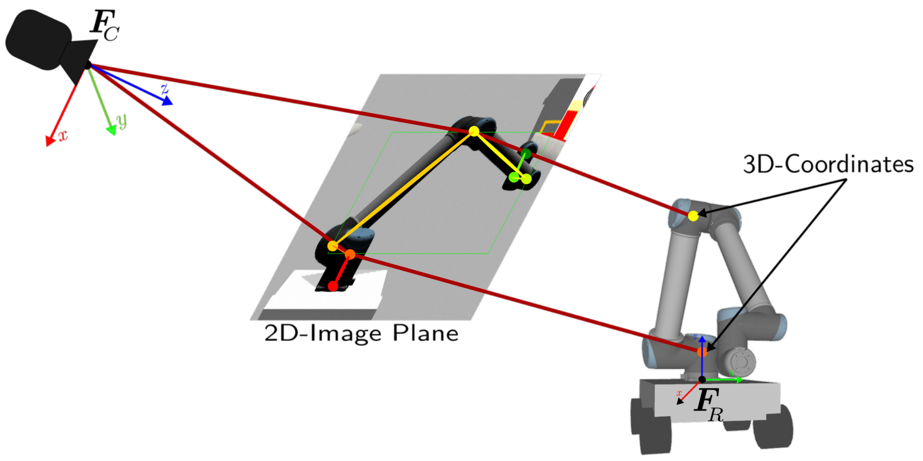


Figure 3.14: Relations between the 2D keypoints and the corresponding 3D positions known by the robot controller (adapted from [I] Fig. 5) © 2018 IEEE.

Ideally, each image of a robot-arm can be used to extract all 2D keypoints of the system which will deliver up to seven unique correspondences (cf. Section 3.3.1). These correspondences can then be fed into any known PnP algorithm (cf. Section 2.4.3) to estimate the camera pose. As the robot's intrinsic poses usually originate in its intrinsic coordinate system, the resulting camera pose will also be relative to the robot base.

As already mentioned in Section 3.3.2.2, the final accuracy of the employed calibration method [103, 193] depends on the accuracy of the extracted 2D keypoints. As it can not be expected that the presented *RoPose* system delivers perfect joint coordinates (see Section 5.1 for a detailed evaluation), an algorithm to collect and organize the estimated joints also were presented in [II]. When it is possible to assume the transformation between the robot's base position and the camera temporarily (or permanently) as static, the correspondences of multiple frames can be collected and stored. In addition to the bare collection of the positional information, the confidence of each joint given by the proposed heatmap from the CNN (cf. Section 3.3.3.2) is used to rate each estimation. Based on this rating it is possible to feed only the best-rated

[12]Disclaimer: The Reutlingen University has a patent pending for the here mentioned calibration procedure and system (DPMA: Nr. 10 2018 101 162.8). The author of this thesis declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

correspondences into the PnP algorithm and improve the overall calibration accuracy [II].

A critical drawback of the presented calibration scenario is that the controller information and the extracted keypoints have to be recorded as synchronous as possible. Especially when the calibration is performed while the robot is rapidly moving, this can lead to a relevant discrepancy between the correspondences which will strongly affect the calibration accuracy. As our final application is realized with ROS all information was synchronized based on its availability within the ROS environment and the contained timestamps[13].

[13]The official supplementary video created for *ICRA2019* also covers the autonomous calibration process and can be found at *IEEEExplore*: https://ieeexplore.ieee.org/document/8793900/media

4 Enhanced Collaborative Workspace Observation

4.1 Simultaneous Human and Robot-arm detection

To detect the pose of humans and industrial robots simultaneously was the logical next step after a sole robot-arm pose estimation was available in order to enable the aspired workspace observation for collaborative systems. As shown in Section 3.3.3.2, a somewhat simple network architecture that was originally developed for human pose estimation was successfully adapted to robot-arms. At this point, there is no need to reinvent the wheel and drop all valuable information already obtained e.g. from the pre-detector that natively contains a class for humans. The proposed architecture for the *ColRoPose* system is shown in Figure 4.1 and consists of two identically structured, but independent, pose heatmap regression networks.

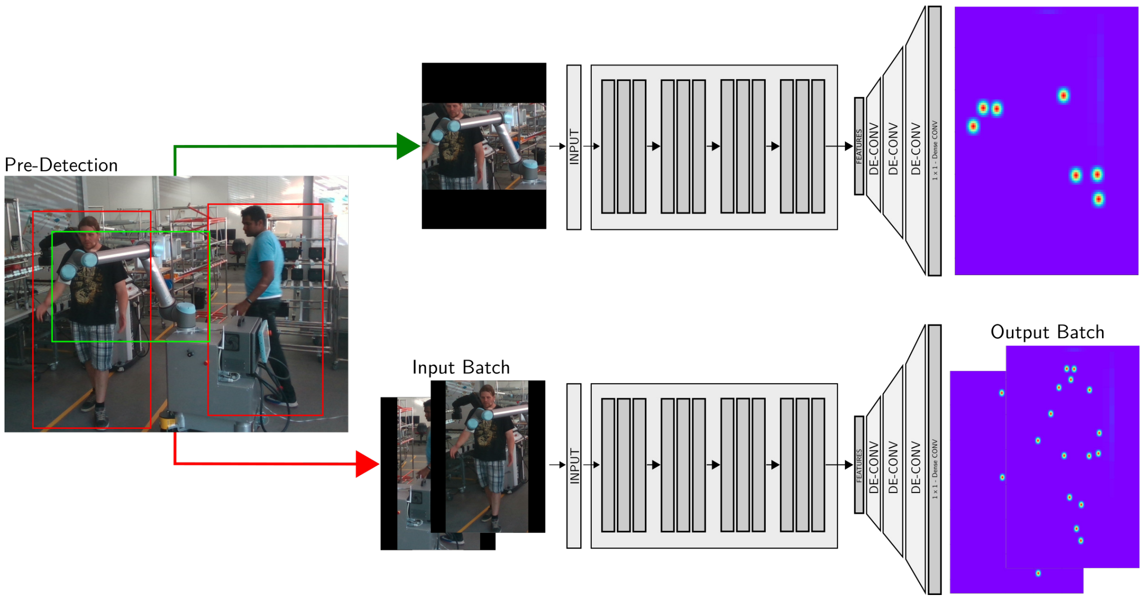


Figure 4.1: System architecture for the collaborative pose estimation system. Both estimation networks follow the same structure and pipeline and are based on the extracted bounding box estimations of the pre-detection system.

It is obvious that this architecture will double the parameter size of the complete estimation model, which could be problematic for the target system, e.g. owed to a possible memory shortage on edge devices. Besides this drawback the separated architecture also has some advantages. One of them is, that both estimation pipelines can be trained independently and on separate datasets. As the robot detection system of *ColRoPose* could be trained on the specific generated datasets as already detailed, the human pose estimation again sticks to the original model [178] and the official model-code from the repository[1] was wrapped in the RoPose project. As *RoPose* is a novel system, it is most likely that the available datasets will grow in the future. This will add more diversity like new robots, new scenes, new poses, and also new special robot appearances to the database and a combined model would require new training or fine-tuning of the whole system, while the realized architecture will restrict the additional training effort to the robot pose estimation network.

Some final estimation results of *ColRoPose* are shown in Figure 4.2. In addition to these images the performance for all multi-person datasets are visualized in a video available online[2]. Please note that the generated collaborative datasets do not contain the GT for the human instances. This has on one side technical issues because no proper tracking system that would be more accurate than visual tracking systems was available within the robots operating space, and also human resource reasons which prevented the research team from label the human poses for the recorded datasets by hand.

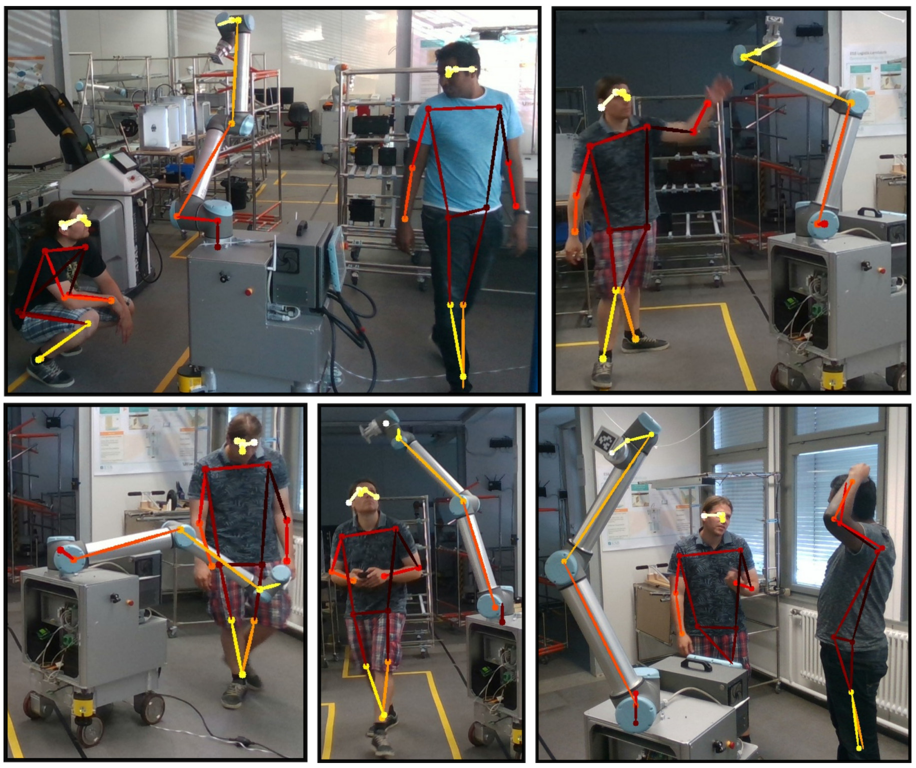


Figure 4.2: Example estimation results of the proposed *ColRoPose* system.

[1] `https://github.com/microsoft/human-pose-estimation.pytorch`
Commit: 18f1d0fa5b5db7fe08de640610f3fdbdbed8fb2f

[2] `https://www.thomas-gulde.de/ropose/result_example`

The missing GT for the human poses is the sole reason why there is no proper evaluation of the human pose estimator on the newly generated datasets and this work will stick to the original authors' evaluation [178]. As shown in Figure 4.2, the system is basically able to reconstruct the poses of multiple human instances as well as of the robot system within the collaborative area. Figure 4.2 also shows the noticed main drawback of the system, namely occlusion. The human detection part can not handle the occlusion from the robot system properly and this is also the case for *RoPose* itself despite it is not shown in Figure 4.2.

The architecture detailed in Figure 4.1 is not the sole solution to tackle the heatmap extraction task and there is a possibility to keep the networks main structure but reduce the over-all parameter size. As the only model base, both pipelines have in common, is the pre-detector, two separate feature extractors, and de-convolutional stacks are needed to perform both regressions. In order to also generalize the feature extractor for both estimation systems, some experiments were conducted. By the time of the submission of this thesis the author was not able to train a more general feature base suitable for both regression systems and the results were not usable for the given purpose. At this point, the author also can not name the exact reason for the occurring problems but expect them to be mainly training and dataset-specific issues. Detailed guesses why the training of a general model failed include:

- **Unbalanced Data**
  Because the datasets like *COCO* [54] are available for years, it is clear that all those datasets are in fact larger, more general, and more diverse than the here presented self-generated *RoPose* datasets. The *RoPose* datasets would need more estimation targets, various backgrounds, light conditions and domains as well as occlusions. This discrepancy leads to unbalanced training especially of the feature extractor.

- **Incompatible feature structure**
  Besides the fact that feature extractors have proven to be able to extract general features of multiple classes, it could always be possible that the given feature-base needed for the regression of the heatmaps for the prediction of the joints of humans and robots are incompatible. In the author's opinion, this is more likely to be not true. Maybe the problem of a more diverse joint estimation system needs to be tackled from another point of view and future applications need to stop to distinguish between human joints and robot joints and need to start to develop a general joint detection system and add an additional joint classification afterward.

- **The need of a special, problem specific, training routine**
  To cover the problems introduced by two different pose models, a very special training routine would be needed which could handle at least the problems with unbalanced data. This will include options like freezing the weights of the feature extractor after some training iterations, or a more or less adaptive training data scheduler which could prevent the model to overfit in one direction by introducing a smart way to arrange the training samples.

Because of these open problems, combined with the lack of any super-computer training system to perform a proper hyper-parameter and data constellation search or optimization [235], the decision was taken to stick to the introduced parallel architecture.

4.2 Kinematic Chain Tracking

Many of the mentioned use cases and applications which can profit from *RoPose* involve continuous supervision of more or less wide areas. Within such situations, it could also become useful if the proposed system would be able to differentiate between various instances of robots and humans within the sensor's field of view. In order to allow for such behavior, each instance needs to be sufficiently described based on the extracted information. Such a descriptor can also be used to distinctly track an instance in a continuous sensor stream. The resulting positional history can then be exploited to roughly predict future locations of the instances to overcome possible errors and miss-detections of other involved systems, like the employed pre-detection system (cf. Section 3.3.5). In order to avoid a tracking solution that is based on the visual appearance of a target which will lead to problems e.g. with identical robot types, the keypoint-based tracking pipeline outlined in the sequence diagram in Figure 4.3 were integrated into *RoPose*.

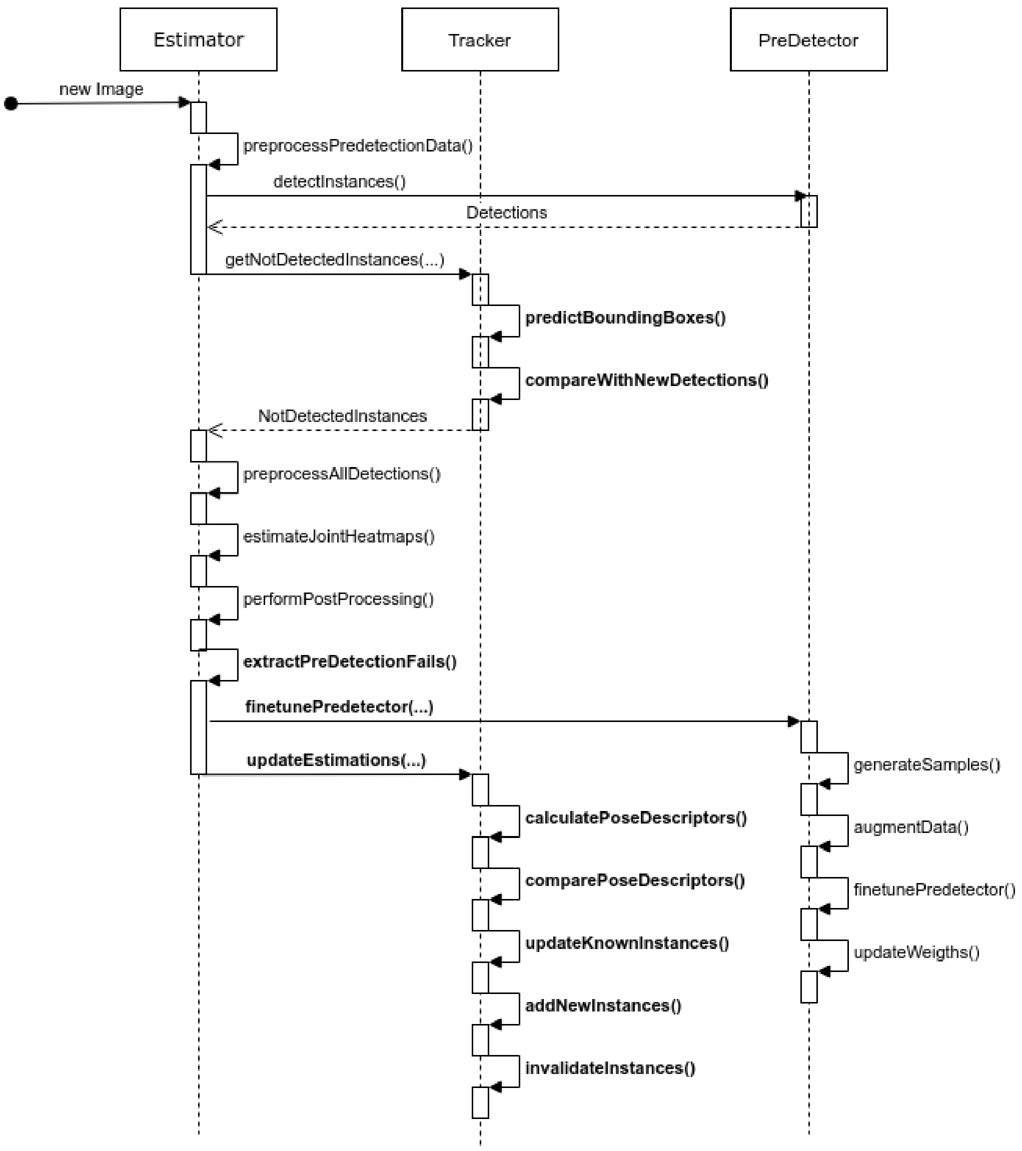


Figure 4.3: Sequence diagram of the proposed tracking algorithm. Bold methods indicate the main parts of the tracking pipeline. The *YOLO* detection model can optionally be finetuned with examples where the pre-detection itself fails but the tracking pipeline does manage it to predict bounding boxes based on the known history of the tracker which lead to successful joint detection of humans or robots.

After a new image is available for the estimation system, all needed steps to perform the pre-detection (cf. Section 3.3.5) have to be performed. The *pre-detector* estimates the bounding boxes for possible targets which are compared to the known bounding boxes stored and organized by the new *tracker*. The *tracker* creates an internal knowledge base of the past system states and is able to predict bounding boxes of already known targets based on their tracked movement history (cf. Section 4.2.3). The predicted bounding boxes are compared to the *pre-detector's* estimations and will return just the predicted bounding boxes that are not covered by the already available detections. These bounding box instances provided by the *tracker* are just added to the *pre-detector's* results and the main *estimator* will use this combination of predicted ROIs to extract a joint estimation, employing the known procedure (cf. Section 3.3.3.2 and Section 4.1). The *tracker* uses the extracted pose information to calculate the current pose descriptors for each instance (cf. following Section 4.2.1). Directly after the final pose estimation the given information can be used to detect ROIs that the *pre-detector* was not able to detect. The additional bounding box information is used to generate new training samples that can be used to autonomously finetune the employed pre-detector as detailed in Section 4.3.

4.2.1 Pose Descriptor

A pose descriptor should offer an abstract way to sufficiently describe the joint constellation of a robot arm. This will allow for a comparison of such descriptors and identify specific instances of a target in different frames which can be used for tracking. Figure 4.4 provides an overview of the elements used to calculate a pose descriptor based on the joint positions extracted with *RoPose*.

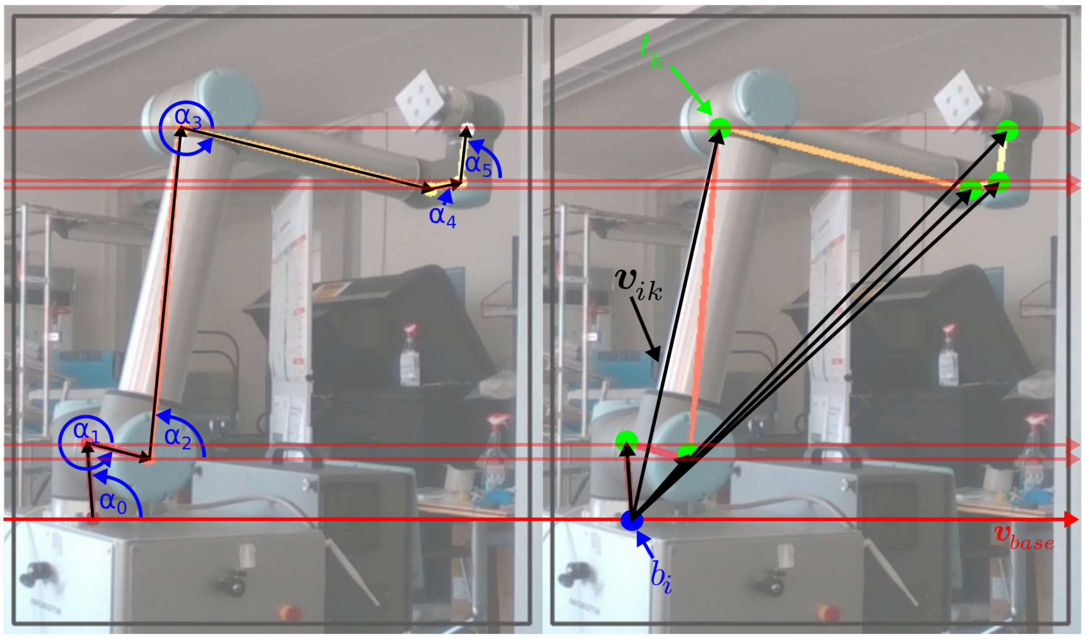


Figure 4.4: Visualization of the features used for pose description. The baseline ($\mathbf{v}_{base}$) is defined by the X-axis direction of the image frame and is used as the comparison orientation for each angle α_n. For each base-joint (b_i) a vector $\mathbf{v}_{ik}$ can be calculated pointing to each estimated target-joint (t_k). The collection of the lengths and angles of the vectors is used as the pose descriptor.

The complete joint-based descriptor itself is defined by a collection of parameter sets ($\mathbf{p}_j$) with a descriptor size (d_{size}) that depends on the number of available joints (j_{max}) of the specific model.

$$d_{size} = j_{max} * (j_{max} - 1) = 42, \; with \; j_{max} = 7 \tag{4.1}$$

The pose descriptor's parameters are based on a set of describing vectors $\mathbf{v}_{ik}$. Each of these vectors can be defined by iterating through all pose joints and calculating the vector between the current base-joint (b_i) and every target joint (t_k) available in the pose model. So basically each base-joint has $j_{max} - 1$ counterparts leading to the in Eq. 4.1 detailed descriptor size. Defining each base-joint as a local origin for the vector $\mathbf{v}_{ik}$ is formally defined as follows:

$$\mathbf{v}_{ik} = \mathbf{t}_k - \mathbf{b}_i \tag{4.2}$$

The baseline vector $\mathbf{v}_{base}$ shown in Figure 4.4 is defined by the X-axis of the image and is used as the reference orientation for the following angle definitions. The parameter vector $\mathbf{p}_j$ consists of two parts encoding the angle α_{ik}, defined as the angle between $\mathbf{v}_{base}$ and $\mathbf{v}_{ik}$ and also the length information of each vector normalized with length of the image diagonal d_i.

$$\begin{aligned} v' &= \mathbf{v}_{base} - \mathbf{v}_{ik} \\ \mathbf{p}_j &= \begin{pmatrix} \alpha_{ik} \\ \frac{|\mathbf{v}_{ik}|}{d_i} \end{pmatrix} = \begin{pmatrix} |\arctan 2(\frac{v'_y}{v'_x})| \\ \frac{|\mathbf{v}_{ik}|}{d_i} \end{pmatrix}, \text{ with } 1 \leq i \leq d_{size} \end{aligned} \tag{4.3}$$

The calculated parameter sets can then be used to derive a similarity (sim_{AB}) between two descriptors, comparing each parameter set p_{jA} from a descriptor A with another descriptor B with the parameter set p_{jB}. Within this similarity calculation, two different normalizations are introduced to enforce a value range of $0.0 \leq sim \leq 1.0$ for both the relative angle and length. As in $\mathbb{R}^2$, the resulting angle difference can be normalized as follows.

$$sim_{\alpha}(p_j) = 1 - \frac{\alpha_A - \alpha_B}{\pi}, \; 0.0 \leq sim_{\alpha} \leq 1.0 \tag{4.4}$$

In contrast to [6] where only joint distances are encoded in the descriptor, the presented similarity calculation also uses the additional angle based joint relations. A more advanced normalization method has to be used for the length, taking the areas into account that are defined by each target's 2D bounding box (b_T) described by the joint constellation of the descriptor's target (T). The bounding box based area norm factor (b_{norm}) is given by the square root of the mean area of both bounding box areas $A(b_A)$ and $A(b_B)$

$$b_{norm} = \sqrt{\frac{A(b_A) + A(b_B)}{2}} \tag{4.5}$$

$$A(b_T) = b_{T,x} b_{T,y} \tag{4.6}$$

and is used for normalization of the length difference between each parameter set that can be used to also calculate the length similarity sim_{len}:

$$sim_{len}(p_j) = 1 - \sqrt{\frac{(|\mathbf{v}_{jA}| - |\mathbf{v}_{jB}|)^2}{bb_{norm}}},\ 0.0 \leq sim_{len} \leq 1.0 \tag{4.7}$$

Finally, both similarities of a parameter set can be combined using the weighted mean of the angle and length similarity. To also enforce the desired value range of $0.0 \leq sim_{jAB} \leq 1.0$ and additional weight constraint have to introduced to keep the summed weight to exactly 1.0.

$$sim_{jAB} = w_{\alpha} sim_{\alpha}(p_j) + w_{len} sim_{len}(p_j),\ 0.0 \leq sim_{jAB} \leq 1.0 \tag{4.8}$$

As these weightings for w_{α} and w_{len} are currently set to 0.5 it is basically the mean value in the current implementation. The overall similarity of the whole parameter set is given by the mean of all similarities:

$$sim'_{AB} = \frac{\sum_{j=0}^{j_{max}} sim_{jAB}}{j_{max}},\ 0.0 \leq sim'_{AB} \leq 1.0 \tag{4.9}$$

To avoid any corruption of the similarity calculation forced by not estimated joints of one of the comparing targets, each sub-similarity calculation where any of these not available joints is involved is ignored and not integrated into the overall similarity calculation. Because the presented descriptor only describes a local pose view, it does not consider the global position of the target on the image plane. This could lead to ambiguous high similarities between multiple targets and an additional global position matching was integrated. For this purpose, the in Section 2.7.2 presented IoU is considered as a good indicator and has been integrated in the similarity calculations, again based on a weighted manner as follows[3]:

$$sim_{AB} = (1 - w_{IoU})sim'_{AB} + w_{IoU}IoU_{AB} \tag{4.10}$$

The presented description method demands that a similarity between two descriptors can only be calculated when a certain number of joints of both targets are available[4]. Besides this missing joint problem, the chosen descriptor method has other drawbacks in the current implementation. The static baseline reference, defined by a specific image locality, and the fact that the descriptor is based directly on the 2D joint projections on the image plane introduces a hard camera viewpoint dependency. However, in order to achieve perspective independence, the extrinsic calibration information (cf. Section 3.4) could be used to project the 2D joints into some kind of a virtual norm space and, respectively, a norm camera plane. A descriptor calculation within such a virtually defined environment could help to eliminate the perspective dependency but is then directly coupled to the extrinsic calibration procedure and the error such a system introduces. It is planned to develop such an advanced descriptor in the future when the calibration error can be reduced. Such a method would also introduce new problems and unknowns, like advanced multi-instance handling, and would require further research and proper evaluation. The currently implemented descriptor and resulting similarities are evaluated in Section 5.6.1 and have been open sourced in [v].

4.2.2 Tracking

With the descriptor specified in Section 4.2.1, it becomes possible to compare derived pose constellations of detected kinematic chains of visible robot arms. The derived similarity between multiple detected targets can then be used to assign instances to each other by matching the targets with the highest similarity considering a defined minimum threshold value[5]. Such a procedure assumes that the movement of a target from one available frame to another will be minimal. The implemented matching itself has to be seen as a greedy algorithm and is performed for all available instances. If there are problems related to very fast movements of subparts of the target, it is also possible

[3]In practice w_{IoU} were set to 0.25.

[4]In practice a similarity between two descriptors is calculated when at least three joint pairs for the comparison targets are available.

[5]In practice and with the used hardware this threshold were set to 0.65 for the given robot targets (cf. Section 5.6.1)

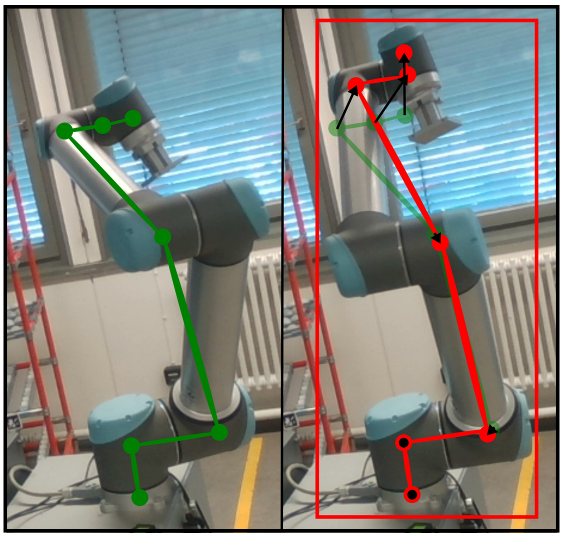
to introduce an additional weight factor for each joint within the kinematic chain. It would probably improve the overall robustness of the descriptor by lowering the similarity impact for the upper joints of a kinematic chain because the final 2D projections of these parts are also affected by the positions of all superior elements caused by the physical nature of a kinematic chain.

4.2.3 Temporal Target ROI Prediction

The tracking system can now be used to predict the position of the robot system based on its given tracking history. Also similar to Xiao *et al.* [178] and many other available works (e.g. [236, 237]) the 2D representation of the collective motion of the observed joints and targets forms the base for the prediction algorithm. This so called motion field therefore describes a target's joint-configuration over multiple frames. For a more understandable explanation, let's assume that an instance can be successfully tracked and assigned over multiple frames which naturally is an essential requisite for the following prediction system.

Based on the tracker, it is possible to generate a continuous position history for the enveloping bounding box as well as for each detected joint element of an instance. For each joint, the motion from one frame to another can be described and parameterized based on the raw 2D positions resulting in a direction vector $(\mathbf{m})$, an absolute distance (d), and also speed (s)[6] (cf. Section 2.5). The motion of a tracked bounding box is based on the motion of each of the four edge points describing the bounding box itself in the same way as for the joint positions. Figure 4.5 visualizes both tracking methods.

[6]The speed parameter is handled on an iterative per-frame basis, no particular time relation was introduced here but could improve the system especially when the frame rate can not be assumed as constant.


(a) Jointwise Tracking

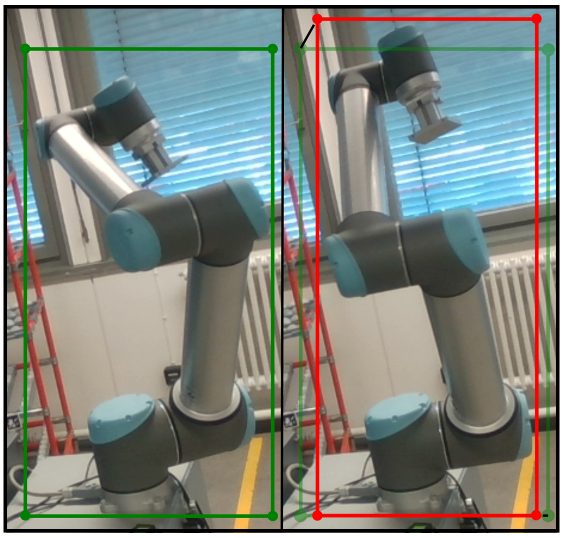

(b) Bounding Box Tracking

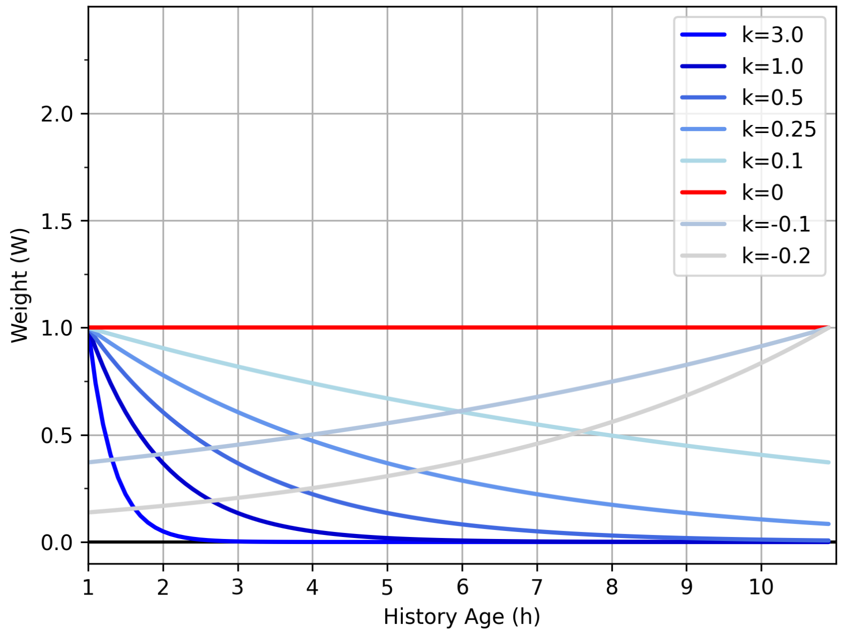
Figure 4.5: Comparison of both tracking methods. (a) Prediction is based on a motion flow tracking of each separate joint. The target's bounding box is subsequently recreated based on the predicted joint positions. (b) Prediction is based on a motion flow of the two defining bounding box corners' positions.

For the pose based tracking, the bounding box needs to be extracted from the predicted kinematic chain, which is performed by finding the bounding coordinates defined by the given joint positions and expanding the resulting box by 20%. Based on a static count of history motion flow data H, each specific history h is also weighted by a given weight w_h to adjust the particular impact on the final prediction. This weighting mechanism should help the tracking system to produce more accurate predictions that are adjusted to the targets performed trajectory. The predicted position $(\mathbf{P}_t)$ for a current frame (t) is then calculated as follows:

$$\mathbf{P}_t = \frac{\mathbf{P}_{t-1} + \sum_{h=1}^{H} \mathbf{m}_h d w_h}{H}, \text{ with } \sum_{h=1}^{H} w_h = 1.0 \tag{4.11}$$

The weight factor was introduced to dampen the influence of older motions, so it is important to keep an eye on the correct order of the given histories H. It is defined that h_1 is the latest known tracked motion of an instance and h_H the oldest.

The presented algorithm allows for multiple ways to determine the history weighting. Within this thesis, an exponential weighting mechanism based on a single decay factor (k) was implemented and is shown in Figure 4.6.



$$W_h(k) = \begin{cases} e^{-kh} & k \geq 0 \\ \frac{e^{-kh}}{e^{-kH}} & k < 0 \end{cases}$$

Figure 4.6: History weight factor used by the tracking mechanism. An exponential decaying weight factor shifts more weight to more recent histories. The factor k of W_h of is used to control the impact of the considered movement histories.

To ensure that the accumulated weight factors will be 1.0 the weights also need to be normalized before they can be used in Eq. 4.11 to fit the defined constraint:

$$w_h = \frac{W_h}{\sum_{h=1}^{H} w_h} \tag{4.12}$$

Examples of predicted bounding boxes for both, the pose based and the bounding box based method, are shown in Figure 4.7, while a final evaluation of the performance gained by the tracking and prediction system as well as the influence of the history handling is presented in Section 5.6.3.

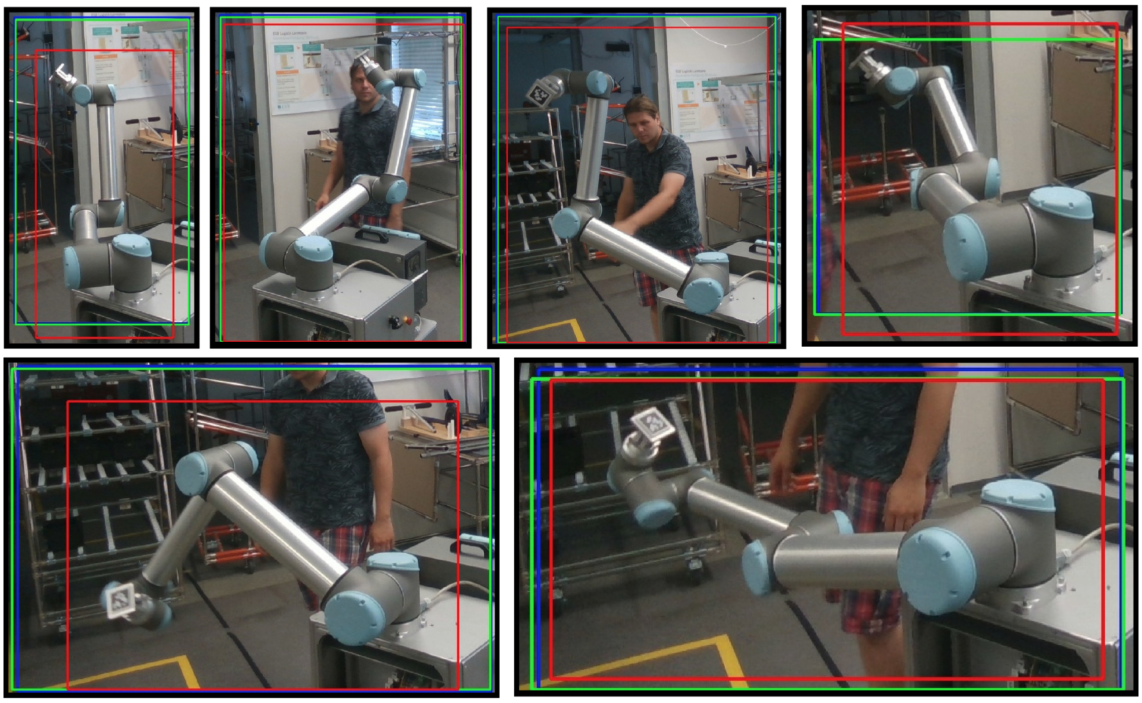


Figure 4.7: Examples of the predicted bounding boxes. The examples combine the GT bounding box (red), the bounding box produced by the pose-based tracker (green) and also the result produced by the bounding box based tracking system (blue)

4.3 Self-Supervised Finetuning of the Pre-Detector

As the pose estimation's accuracy of the basic system without the additional tracking mechanism directly depends on the information delivered by the pre-detector, a system that automatically improves the bounding box prediction model would be helpful. As already delineated in the diagram in Figure 4.3, successful pose detection based on reconstructed tracked bounding boxes in which the pre-detection mechanism failed can be used to generate new training samples to fine-tune the model. Besides normal false negative responses of the pre-detector, occlusion can be seen as one of the main sources of failure where the applied tracking mechanism can help to improve the accuracy. Figure 4.8 shows some examples of automatically extracted fine-tune samples, based on the position information recovered by the tracking.

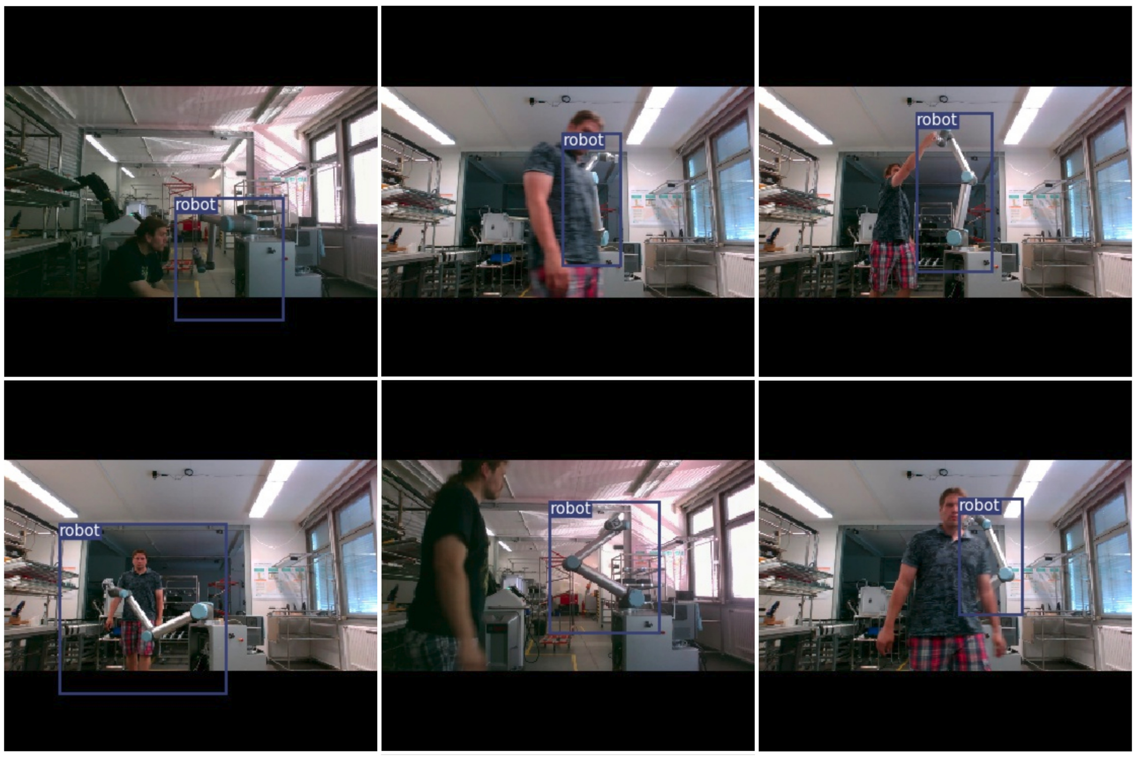


Figure 4.8: Examples of auto-generated training data based on the proposed tracking system. The examples are recreated if the origin pre-detector could not detect an already known target. Such data can be used to finetune a bounding box detector. The produced ROIs could also exceed the shape of the original image. For the fine-tuning process, such predicted areas will be clipped to the given shape of the new training image.

At this point, it is important to understand that there is no out-of-the-box solution for the finetuning of a SOTA deep learning model. Each model has its *mannerisms* and needs to be handled differently in the finetuning process. As it is always a good advice to use small learning rates and usually non-adaptive optimizers like *SGD*, the main challenge is to set up the model's layers. It is rarely a good idea to finetune all layers of a model, as some weights of the network's architecture could be frozen due to the training process [238, 239]. In the case of *YOLOv3*, every layer except the main *YOLO-Layers*, where the network performs the explicit extraction of the anchor boxes on different image scales were frozen (cf. [231, 232, 230]). Precisely these are the convolutional layers 82, 94, and 106 out of the 106-layered architecture. This was also the only direct network modification that was applied while finetuning. For the optimization, the *SGD* optimizer with a static learning rate of $lr_{static} = 1e^{-5}$ was employed.

The proposed self-supervised finetuning should theoretically be able to raise the accuracy of the bounding box detection close to the system with activated tracking. A detailed evaluation of the impact of the presented on-the-fly finetuning process can be found within the evaluation in Section 5.7.

Despite the fact that the basic evaluation shows that the accuracy of the bounding box detector can be improved with such a mechanism, potential users have to be conscious of over-fitting the model to the given domain of the finetuning process. As such an over-fitting could not lead to noticeable problems in applications where e.g. the specific model is only targeted in a well defined and constant area, it is preferable to train on as general data as possible. To avoid over-fitting it is recommended to mix the self supervised training's examples with already known data used for the bootstrapping learning process of the model.

5 Evaluation

5.1 Keypoint Detection

For the whole *RoPose* related evaluation presented in the following sections, the same amount of evaluation data was used and none of the systems were ever trained with these datasets. This so-called *hold-out* validation is a suitable method to evaluate the generalized performance of a system, but also has some drawbacks such as the loss of valuable evaluation data for the training process (cf. [240]).

The evaluation data collection consists of twelve separate *RoPose* datasets generated as detailed in Section 3.3.2, summing up to exactly 3213 evaluation samples with five different camera positions relative to the robots origin. The datasets containing labeled samples, with and without additional single or multiple humans[1] also introducing partly or complete occlusions of the robot itself.

A complete iterative *k-fold* validation (cf. [241, 240]) was not performed mainly because of the given time and hardware constraints. An e.g. ten-fold cross-validation would have required ten complete training sessions for each of the presented networks, which would sum up to over a month of training time with the available hardware setup. However, the current evaluation codebase would also need additional dataset scheduling algorithms to perform such validations, which is surely on the agenda of the author's future work.

5.2 PCK Metric

Before presenting the results of the evaluation, a relevant metric is briefly introduced. As the PCK metric was discussed in Section 2.7.3 the threshold factor (f_{th}) needs to be defined. The factor f_{th} could be related to the length of the Bounding Box (BB) as proposed by Yang *et al.* [155], something like the head segment of a human [156] or it could also be set to a static value. While a robot's equivalent to a human's head segment could be the distance between the first two known joints (cf. Figure 3.1), this distance is used as the possible dynamic reference value. Table 5.1 gives an overview about how

[1]Note that the available labels not contain human pose transformations yet. Only the poses of the robot's kinematic chain are available within the collected data.

these different approaches would affect the absolute threshold reference based on the available *RoPose* datasets.

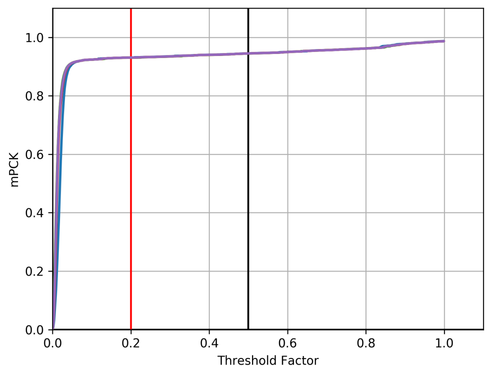
Table 5.1: Possible PCK threshold references lengths.

	Mean	**Max**	**Min**	**Mean**	**Max**	**Min**
Base Seg.	36.6	51.8	31.7	49.6	73.1	28.2
BB Diag.	512.8	1335.5	104.1	692.5	1497.2	104.1
BB Height	394.5	719.0	58.8	522.0	719.0	58.8
BB Width	321.9	1125.4	85.8	435.2	1279.0	85.8
	Evaluation Datasets			All Datasets		

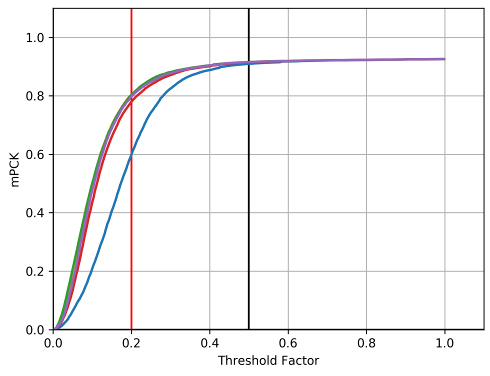
Absolute dimensions of the possible PCK threshold relations in Pixel [pix]. The numbers show the mean, maximum and minimal lengths for the different approaches to calculate the threshold reference values for all available real datasets (training-, test- and evaluation Dataset) and also for the evaluation datasets only.

The numbers show that all BB relative values introduce a very low error rate in higher threshold factors. As suggested in Section 2.7.3, the PCK could be evaluated based at a very low factor just as 0.1 which could also be used for the here presented evaluation [155]. Figure 5.1 shows the example PCK plots based on the introduced threshold references, evaluated at various factors.

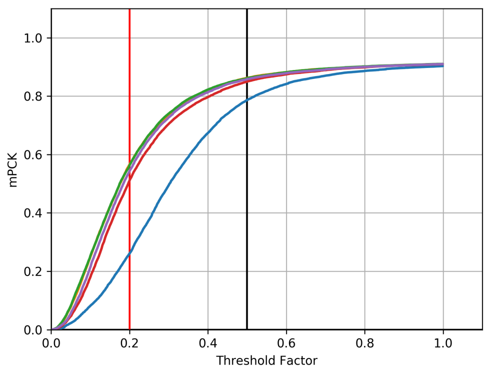
It is obvious that a f_{th} relative to the diagonal length of a bounding box (Figure 5.1(a)) adds almost no value to the evaluation because the resulting absolute steps between the thresholds are very large, and the steep slope will not allow for a proper comparison of the curves. As the base segment length (Figure 5.1(b)) also offers a way to compare the performances especially at lower absolute thresholds, this target size depending view is used whenever a PCK result of an estimator is compared. It would also be suitable to set the relative factor to a static number such as 20 (shown in Figure 5.1(c)). Because the evaluation will also provide an absolute comparison of the given predictions and the GT on the pixel level, it is considered as not necessary to cover this case when the PCK results are evaluated.



(a) Threshold reference set to the GT's BB diagonal.



(b) Threshold reference set to the base segment's length.



(c) Static threshold of 20 pixel.

Figure 5.1: Effect of the relative thresholds to the PCK metric. (a) f_{th} relative to the GT's bounding box diagonal length. (b) f_{th} relative to the GT's base segment length (length between the first two joints starting with the robot's base). (c) f_{th} relative to a static pixel value (example: 20pix). The colors indicate results for each joint.

5.3 Bounding Box Detector

To evaluate the *YOLO* based bounding box detection system (cf. Section 3.3.3.2 and 3.3.5), the IoU-Metric (cf. Section 2.7.2) serves as the main key-value in the following evaluation.

The IoU of the predicted bounding box and the the corresponding GT were evaluated with a varying threshold factor (th_{bb}, with $0.0 <= th_{bb} <= 1.0$). In this case, th_{bb} defines the rejection threshold that is used to classify all detections where $IoU < th_{bb}$ as not detected. This methods results in a basic binary classification and the *Precision and Recall* metrics (cf. Section 2.7.1) can be finally applied. The trained *YOLO*-model's performance for the robot bounding box detection is summarized in the Precision/Recall-curve in Figure 5.2.

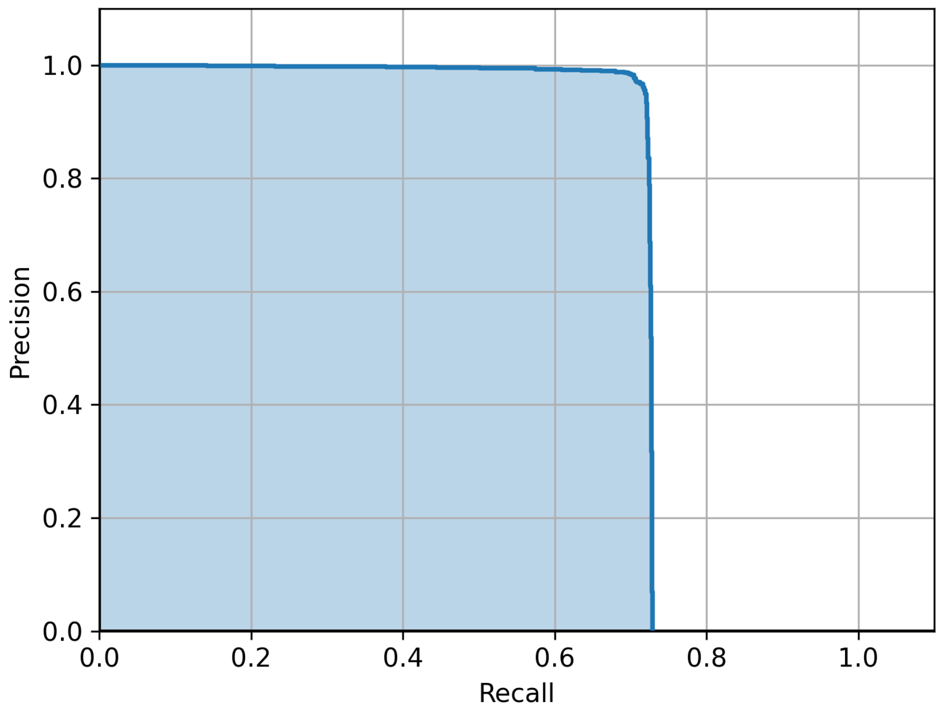


Figure 5.2: PR-curve of the YOLO-based *RoPose* bounding box detection. The AUC results in an AP of ~0.724. Details of the employed metric are given in Section 2.7.

The reached F1-Scores of the detector evaluated at the given thresholds are shown in Figure 5.3.

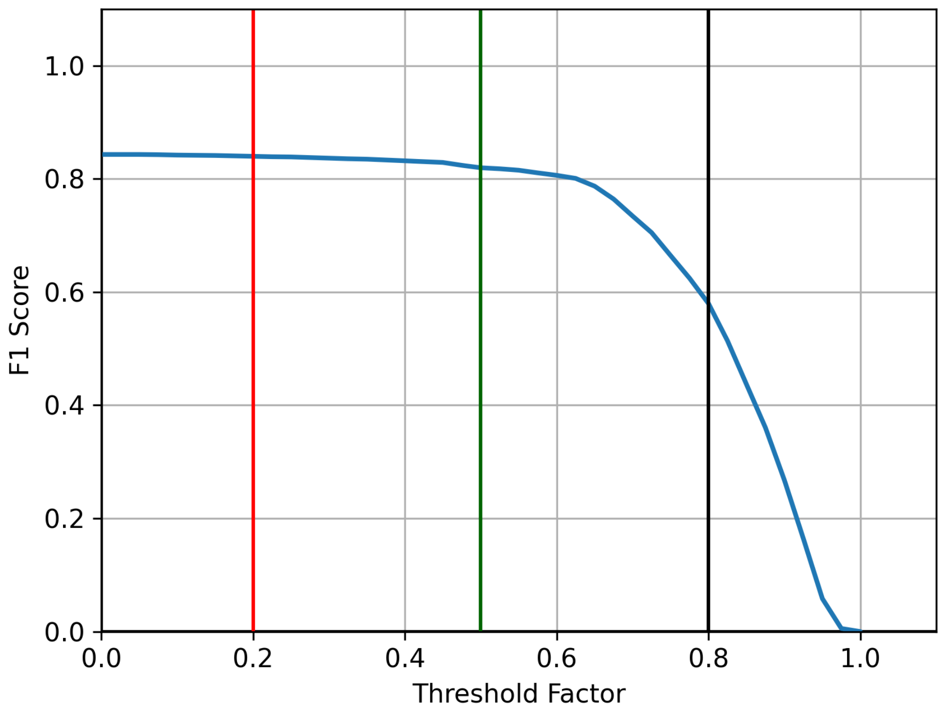


Figure 5.3: F1-Scores of the YOLO-based *RoPose* bounding box detection evaluated at various thresholds th_{bb}. Details of the employed metric are given in Section 2.7.

The evaluation procedure enables the extraction of the key values listed in Table 5.2 and allow for an objective rating of the employed detector's final performance on the evaluation datasets.

Table 5.2: Evaluation key values of the bounding box detector.

	AP	**F1@0.2**	**F1@0.5**	**F1@0.8**	**mIoU**
Results	0.724	0.84	0.82	0.58	0.418

The resulting AP and F1-Scores at the evaluated thresholds as well as the solid mean IoU of the *YOLO*-based bounding box extraction shows that the system performs well enough to serve as the needed pre-detection in the given data domain because the detector is able to detect slightly over 80% of the target robot arms with an IoU above or equal to 0.5. The comparable low mean IoU can be traced back to FN detections. Such FNs will directly lead to a zero IoU and have a high impact on the mean value.

5.4 RoPose Base System

After the pre-detection, the *RoPose* keypoint estimation system needs to be evaluated. As detailed in Section 5.5, it is recommended to include an upsampling of the probability map to increase the final accuracy. For the following evaluation an additional upsampling step that increases the 64×48 sized network outputs to the origin input size of 256×192 based on an bicubic algorithm was employed (cf. method D in Section 5.5). To train the system the routine described in Section 3.3.3.4 was employed. The resulting PCKs for each of the joints are shown in Figure 5.4.

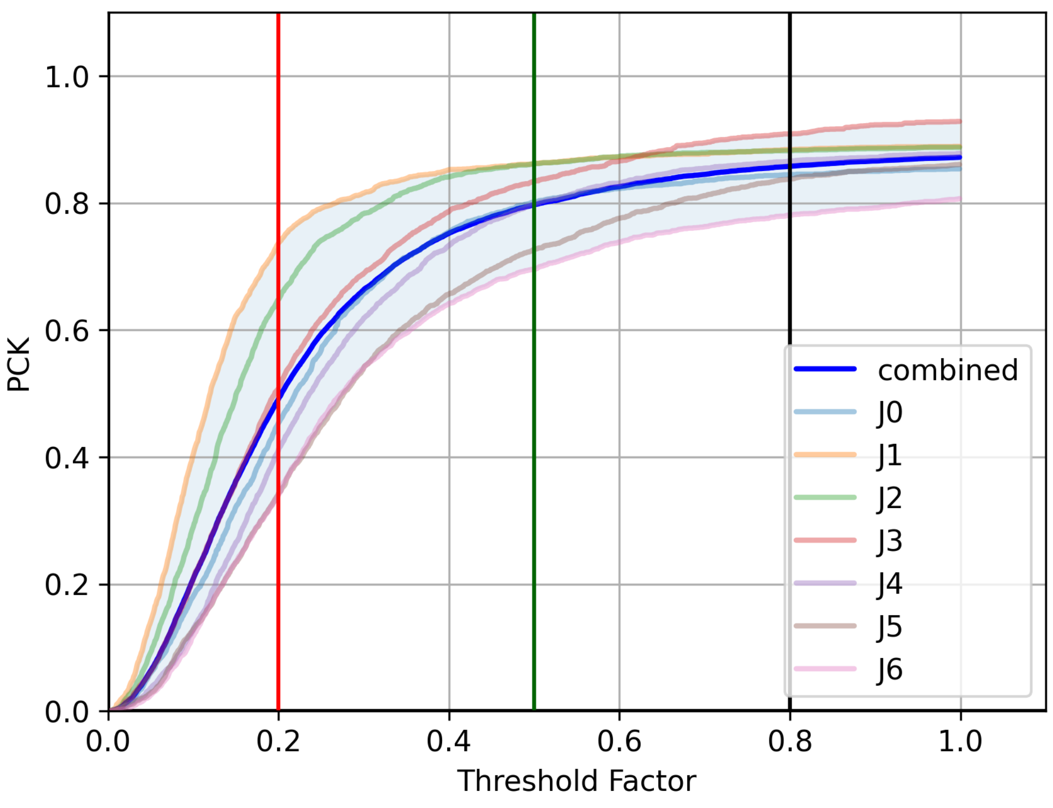


Figure 5.4: Joint estimation PCK results. While the combined value (blue) encodes the mean of all joints, each joint's PCK is shown in the respective color. Details of the employed metric are given in Section 5.2.

The development of the PCK-curves is almost identical and thus, there is some variance between each joint's absolute accuracy, especially when looking at the joints at the end of the kinematic chain. This is the exact same behavior already discovered with the previous published models [I, II] and is likely related to the larger positional variance the upper joints could engage because each joint which can be seen as a lower chain element will affect the final position of such a specific joint.

Table 5.3: Evaluation key values of the base *RoPose* keypoint estimation.

	PCK@0.2	**PCK@0.5**	**PCK@0.8**
combined	0.498	0.798	0.858
J0	0.449	0.8	0.843
J1	0.736	0.863	0.884
J2	0.656	0.862	0.883
J3	0.521	0.837	0.91
J4	0.425	0.801	0.866
J5	0.349	0.725	0.837
J6	0.346	0.698	0.78

An absolute, not threshold-related, view on an explicit two-dimensional error distance within the given image plane should also be provided when evaluating a system like *RoPose*. These numbers are visualized in the form of violin plots in Figure 5.5, confirming the conclusion drawn from the PCK analysis.

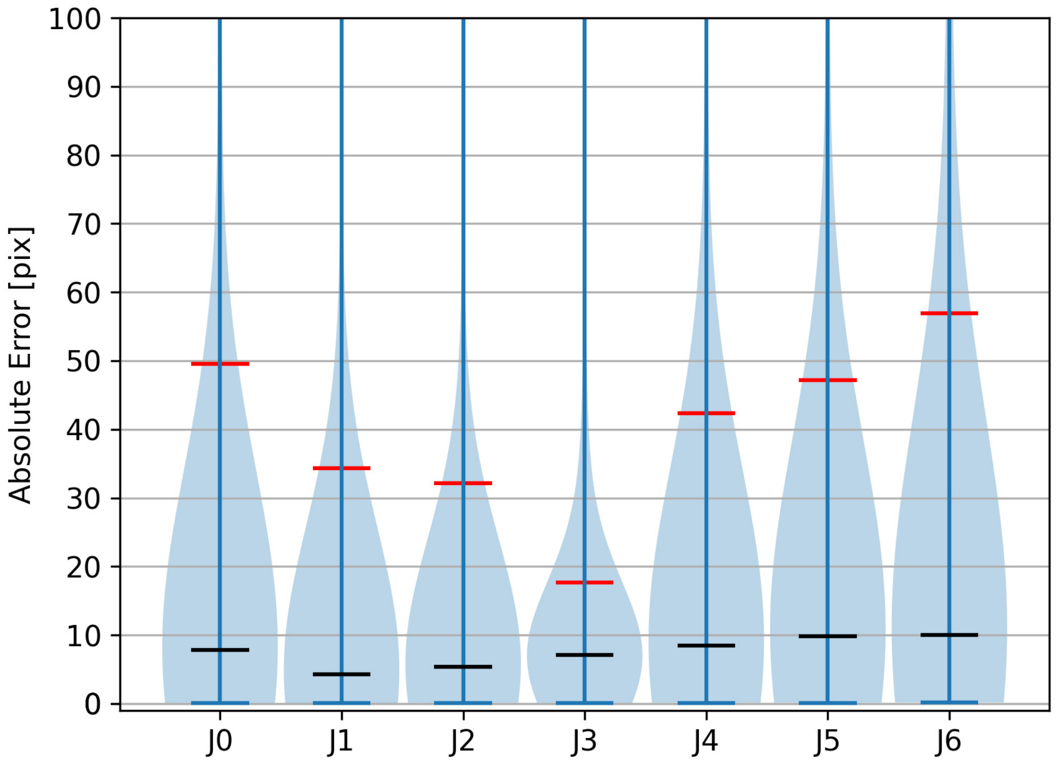


Figure 5.5: Violin graph of the absolute joint-wise estimation error in pixel, evaluated on images with a size of 1280 × 720. While the width of each violin indicates the distribution, the red and black lines denote the mean and median values, respectively.

Again, the data show a lower overall performance of the upper-level joint estimation compared with the first parts of the kinematic chain. It is always important to consider the pixel dimensions of the error. The evaluation was performed on images with a raw input size of 1280×720, resulting in a maximum error distance of $\sim 1468pix$. The comparatively large mean values (red lines in Figure 5.5) occur because the raw estimation system produces some heavy outliers ($e_{max} = 1121pix$). Such outliers can easily be rejected in the final application (cf. Section 6) by comparing the estimated poses with the predicted poses delivered by the tracker. Table 5.3 finally shows the raw numbers of the evaluated key values. Table 5.4 presents the absolute numbers of the given joint-wise estimation errors.

Table 5.4: Rounded absolute joint-wise estimation errors [pix].

	Comb.	**J0**	**J1**	**J2**	**J3**	**J4**	**J5**	**J6**
Mean	39.98	49.68	34.16	32.0	17.51	42.27	47.2	57.05
Median	7.19	7.86	4.19	5.18	6.93	8.22	9.7	9.9
Variance	10.75	11.27	10.15	9.44	7.09	11.15	11.52	12.2

The evaluation of the main keypoint estimation shows that the system performs sufficiently well on the given data basis. The data show that the developed model can be used within the proposed system architecture to perform the keypoint regression, and combined with the post-processing pipeline it will produce accurate positional guesses. However, this evaluation is performed on recorded datasets where the base of the robot is static within the domain of each set. It is part of the authors future work to realize a technical solution to also generate datasets with a dynamic moving robot platform. Such an addition will improve the system's performance and especially the quality of available datasets.

Because the above evaluation was based on the perfect bounding box input given by the GT, it is also necessary to investigate how the system behaves based on the bounding box information from the pre-detector. As these evaluations can be perfectly coupled with a direct comparison of the tracking, this evaluation is also placed in Section 5.6.3.

5.5 Output Upsampling

In order to show the impact of the different upsampling strategies (cf. Section 3.3.4) of the introduced probability maps, an evaluation of the joint estimation performance was performed for each of the available upsampling methods (*None*, *Bilinear* and *Bicubic*) and also for two different target size strategies. As the size of 256×192 is the input size of the heatmap regression network, the term *Original Size* specifies a direct upsampling to the original raw size of the cropped ROI of the estimation target. The main results are summarized in Table 5.5 and show the resulting PCKs of all joints combined.

Table 5.5: Impact on combined PCK of different upsampling strategies.

	Method	**Target Size**	**PCK@0.2**	**PCK@0.5**	**PCK@0.8**
A	None	64×48	0.355	0.774	0.851
B	Bilinear	256×192	0.477	0.797	0.858
C	Bilinear	Original Size	0.462	0.796	0.857
D	Bicubic	256×192	0.492	0.797	0.858
E	Bicubic	Original Size	0.498	0.798	0.858

Comparison of different upsampling strategies and their impact on the pose estimation accuracy. As the term *Target Size* refers to the final target size each probability map was upsampled to, *Original Size* refers to the the raw size of the extracted ROI before it was fed in the pose regression network.

Besides the various upsampling algorithms, Table 5.5 also shows the impact of upsampling the data to the original input size of the cropped robot. The size of the raw ROIs is usually larger than the statically defined input size of the network. Whenever this is not the case, downsampling is performed with the given method. Figure 5.6 shows the combined joint PCKs in more detail.

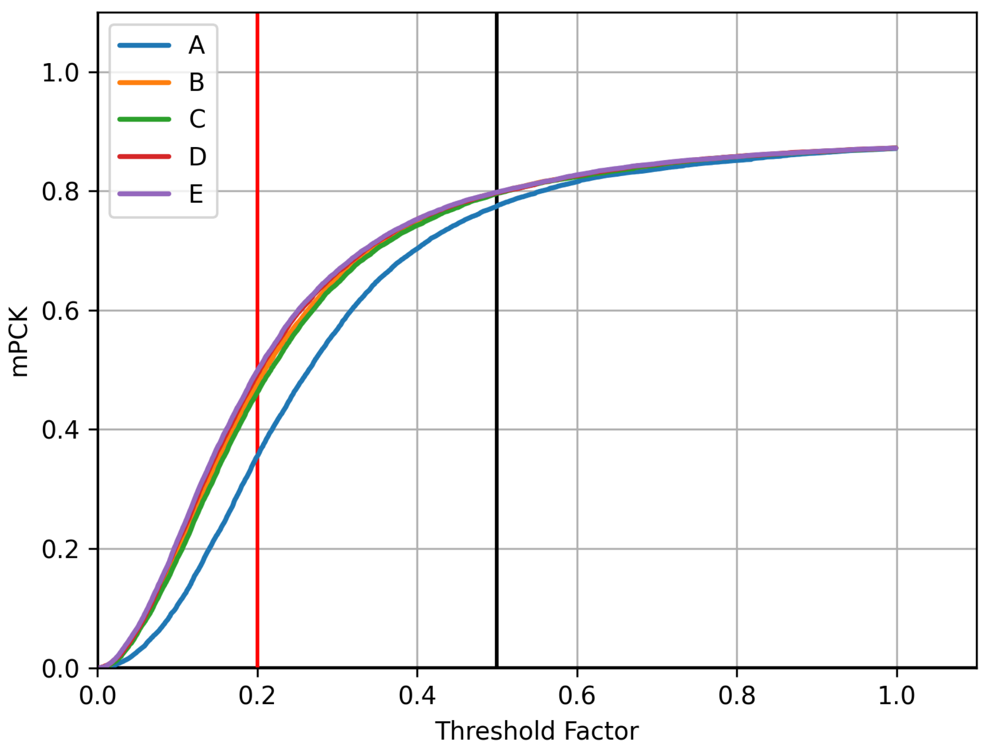


Figure 5.6: PCKs related to different probability map upsampling strategies. All lines show the development of the PCK accuracy, again evaluated at different thresholds. The PCKs combine all given joints of the robot and the enumeration in the legend correspond to the defined characters of each method in Table 5.5.

The results clearly show that the integration of any additional upsampling step (B-E) outperforms the none upsampling strategy (A) when it comes to more restricting thresholds. If any application needs such an accuracy boost, and the time constraints allow for additional processing time, such an upsampling should be considered. As Figure 5.6 shows, there is not much difference for each of the other methods. Setting the target size to the original ROI (B and D) except the original image size (C and E) even performs slightly better when bilinear upsampling is involved but is responsible for a small accuracy decrease with the bicubic setup. The differences between bilinear and bicubic are marginal at high restricting thresholds but may are negligible at systems that do not need this small accuracy boost.

Again the absolute pixel-error has to be considered. Figure 5.7 shows the absolute combined pixel distances for each of the employed upsampling methods.

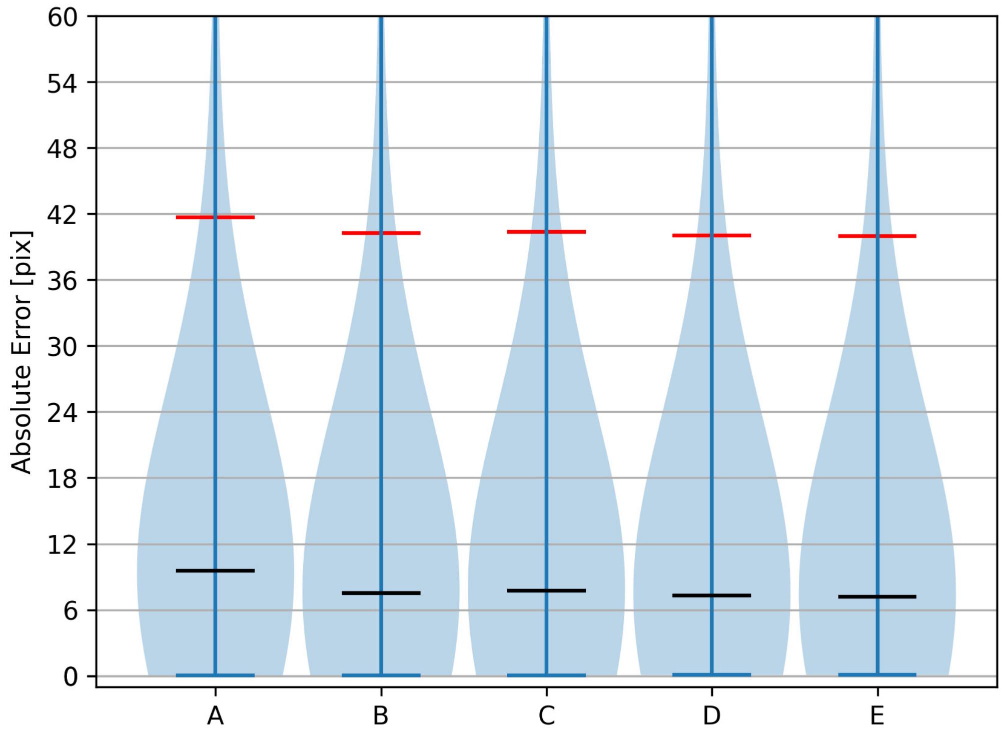


Figure 5.7: Absolute combined error for the upsampling strategies. The violin shows the combined estimation error in pixel, evaluated on images with a size of 1280×720. While the width of each violin indicates the distribution, the red and black lines denote the mean and median values.

The absolute view on the error numbers also confirms the marginal difference between the upsampling strategies. Again the original image's input size of 1280×720 needs to be considered when looking at the numbers presented in Table 5.6 (cf. Section 5.4) .

Table 5.6: Absolute estimation errors of different upsampling strategies [pix].

	A	**B**	**C**	**D**	**E**
mean	41.664	40.266	40.361	40.024	**39.982**
median	9.532	7.527	7.741	7.277	**7.192**

Because the differences between the methods are marginal it was decided to integrate method D, the bicubic upsampling to the target's input size, in the final *RoPose* application. This decision was mainly based on a computational point of view as the strict target size of 256×192 will reduce the computation effort of the keypoint extraction to a more deterministic value as compared to a variable output size coupled at the raw image input's ROI.

5.6 Tracking

The evaluation of the tracking system itself is split in three parts. While the Sections 5.6.1 and 5.6.2 examine the performance of the base descriptor and the influence of the introduced weighting, Section 5.6.3 will investigate the impact of the complete tracking procedure. As already detailed in Section 4.2.3, the tracking procedure offers two different methods where the tracking is based purely on edges of the bounding box (further referred to as method *B*) or all the available poses of the robot's kinematic chain (further referred to as method *A*).

5.6.1 Descriptor

Before it is possible to tell if a tracking mechanism can be applied to a system, the employed descriptor, in this case, the abstract pose based descriptor presented in Section 4.2.1 has to be evaluated. The following evaluations are based on the given GT data of all available datasets, used for training and also for validation. All these data can be used because no training is involved in order to calculate the descriptor and this leads to an overall amount of 29.236 usable data samples.

Each pose descriptor can be represented as a vector containing a parameter set ($\mathbf{p}$) of all 42 parameters (cf. Eq. 4.1 - 4.3). To get an idea how these real data vectors built up a descriptor space, a method called Stochastic Neighbor Embedding (SNE), originally introduced by Hinton *et al.* [242], was employed. The idea behind SNE is to reduce the complexity of higher dimensional spaces and its members and transforms them into a lower dimensional space (e.g. two-dimensional) where it could be more easy to visualize and understand the data. Usually this can't be done by a simple projection and the employed transformation needs to preserve the original data clustering of the multidimensional space. SNE algorithms first assign a random position in the targeted space for each high-dimensional datapoint and fit these positions in an iterative process. Each iteration adjusts the lower dimension positions of the datapoints and move them closer to other samples which have a high similarity in the high-dimensional space. These pairwise similarities between all datapoints are usually modeled by a normal distribution based on the multi-dimensional vector distance of each element. The very same similarity estimation is also applied to the low-dimensional datapoints allowing to fit the data to match the similarities in both spaces. To generate the following plot in Figure 5.8, an extended algorithm called *t-SNE* presented by van der Maaten *et al.* [243] and the publicly available implementation within the *scikit-learn* python toolkit [244] have been used to apply the dimension reduction. *t-SNE* uses *William Sealy Gosset*'s *Student's t-distribution* [245] for the similarity estimation of the lower-dimensional datapoints instead of a normal distribution like introduced in the original work. This helps to better identify data clusters because the similarities will more spread and the high values will not concentrate at the distribution's peak. Because the *t-SNE* algorithm can just operate on numerical vectors, the used data needs to be cleared from all samples that contain not visible joints (cf. Section 4.2.1). For the analysis shown in Figure 5.8 the available 29.236 samples have been cleared from 1.314 descriptor elements that contain invisible joints, leaving 27.922 samples for the embedding[2].

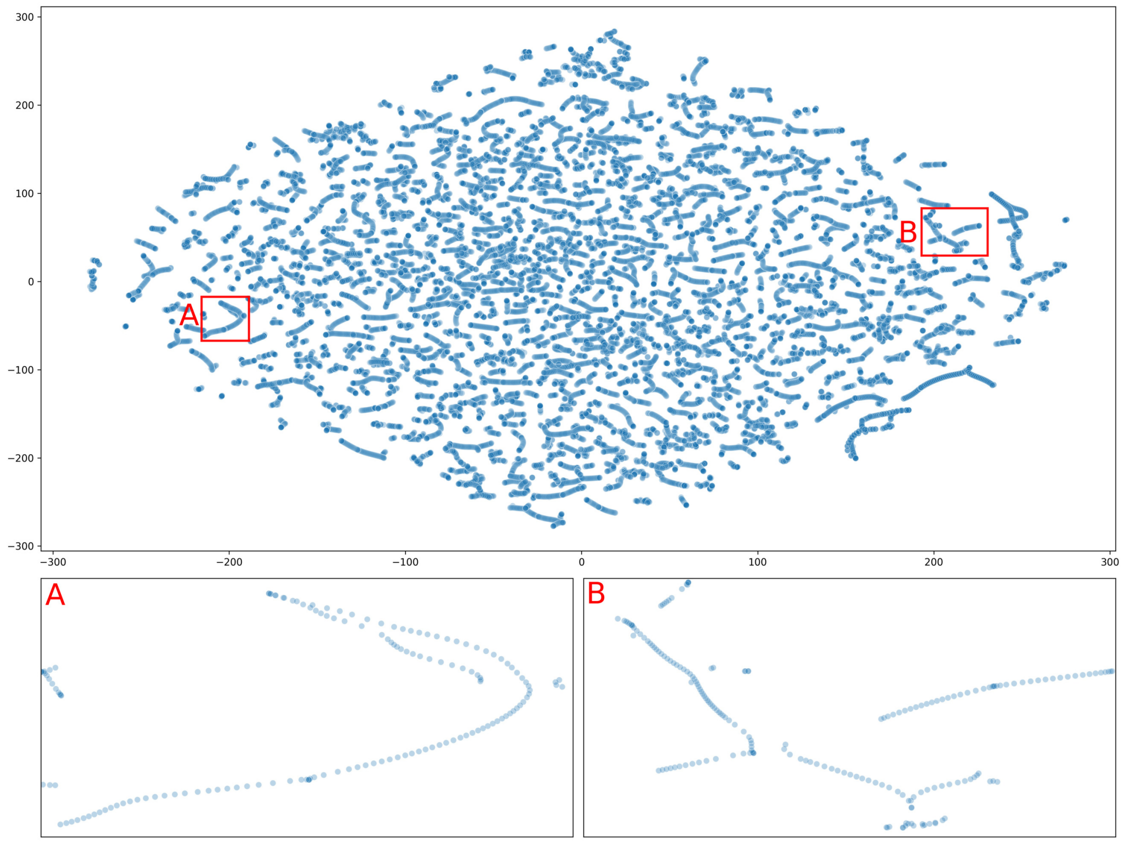


Figure 5.8: 2D-Visualization of the descriptor's feature space. The high-dimensional vector-space of the *RoPose*-descriptor was transformed into an 2D space to visualize the data structure of the feature space by employing *t-SNE* [243]. For the embedding all descriptors with invisible joints have been removed leaving 27.922 samples for the process. A and B show detailed sections from the whole embedded data. These selections show how the introduced descriptor space directly models trajectories into close datapoints as a trajectory usually consists of various similar poses.

The with *t-SNE* embedded feature space visualized in Figure 5.8 clearly shows that most of the descriptors can be clustered into the trajectories that the targeted robot

[2]The t-SNE's perplexity-parameter was set to 10 and a maximum of 5000 iterations for the fitting process were allowed.

executed while recording the datasets for *RoPose*. The data in figure 5.8 show that the introduced descriptor forms a continuous parameter space that can be used to distinguish between poses. To show the embedding of the descriptor space on a more concrete example, a video was created showing the embedding process of the descriptors from a single dataset and the *evolving* trajectories in the embedded features space[3].

The second part of this evaluation targets the usage of the descriptor to estimate the similarity between two kinematic chains and is performed by calculating the similarity between the given robot pose from frame t and $t+1$. Some of the datasets contain frames in which the robot does not move from frame to frame and this means that a similarity value of 1.0 can possibly be reached. Because the GT data has to be considered as *perfect* in the given context, an additional 2D joint noise was introduced. This joint offset was generated for each joint of the kinematic chain in each frame independently and is modeled as 2D Gaussian distribution with $\sigma = 0.5$ and was also scaled and clipped to a range between -40 and 40 pixel. This basically introduces a spatial shift for each available joint on the image plane and the error-range is comparable to the absolute estimation errors of the estimator (cf. Section 5.4). In addition to the displacing, also potentially not detected joints have to be considered. Caused by situations as occlusion, the resulting incomplete kinematic chains will also have an impact on the similarity calculations. For this reason, joints were randomly set to invisible, so the descriptor will ignore this particular joint (cf. Section 4.2.1). This invalidation of joints was introduced with a probability of 0.1. Figure 5.9 shows the histograms of the so calculated frame-by-frame similarities across the whole data basis.

[3]`https://www.thomas-gulde.de/ropose/tracking`

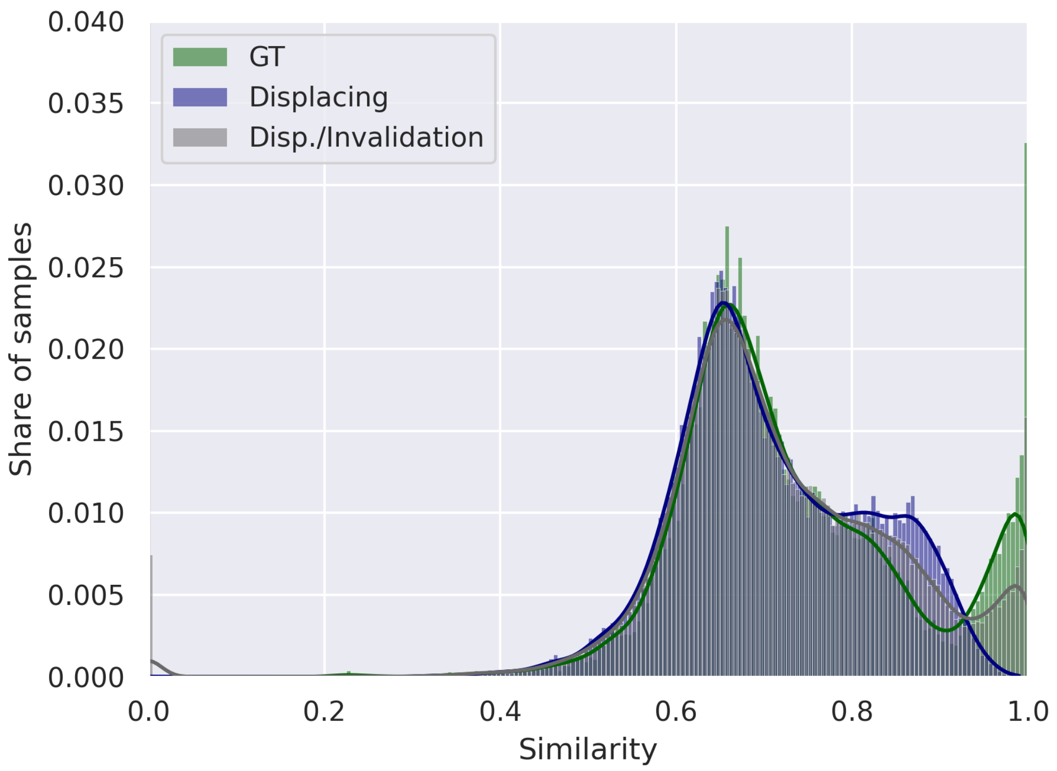


Figure 5.9: Distribution over similarity values of the compared descriptors. The similarities were calculated from the sequential frame information given by the GT of all available datasets. The bounding box weight factor is set to $w_{IoU} = 0.25$ (cf. Eq. 4.10). To add a more realistic noise to the data, each joint was displaced by a 2D Gaussian distributed offset with $\sigma = 0.5$, which also was scaled and clipped to a range between -40 and -40 pixel (blue distribution). The gray distribution shows the results when random joints were also set to invisible by a probability of 0.1.

Besides the direct comparison with the GT (green distribution), Figure 5.9 also shows the similarity results after the virtual displacing (blue curve) and the similarity after the additional joint invalidation (gray curve). These similarity distributions clearly show that the defined descriptor can be used to estimate a similarity between two 2D kinematic chains, which makes the descriptor a possible candidate for the needed tracking pipeline. The marginally positive difference of the joint invalidation manifests due to the invalid joints not being part of the similarity matching (cf. Section 4.2.1), which could lead to an overall greater similarity. The distributions show that a similarity threshold value of around $0.45 - 0.6$ can be considered as a suitable base to match two descriptors. The graph also shows some outliers with a similarity of zero for displaced and invalidated samples. These outliers are caused by to many random joint invalidation which leads to a zero similarity which is the expected behavior.

5.6.2 Influence of the Tracking History

The tracking system and the future bounding box prediction system employ a weighting factor which individually weighs the influence of each known history according to its update time (cf. Section 4.2.3 resp. Equations 4.11 and 4.12). The evaluation of the employed exponential factor based on a total history amount of ten samples used for the prediction allows for conclusions about the importance of older histories. Figure 5.10 shows the impact on the predictions' mean joint distances for different weight factors k (cf. Figure 4.6)for the pose tracking mechanism. This evaluation is again based on all 29.236 samples because no learning is involved for the evaluated system parts.

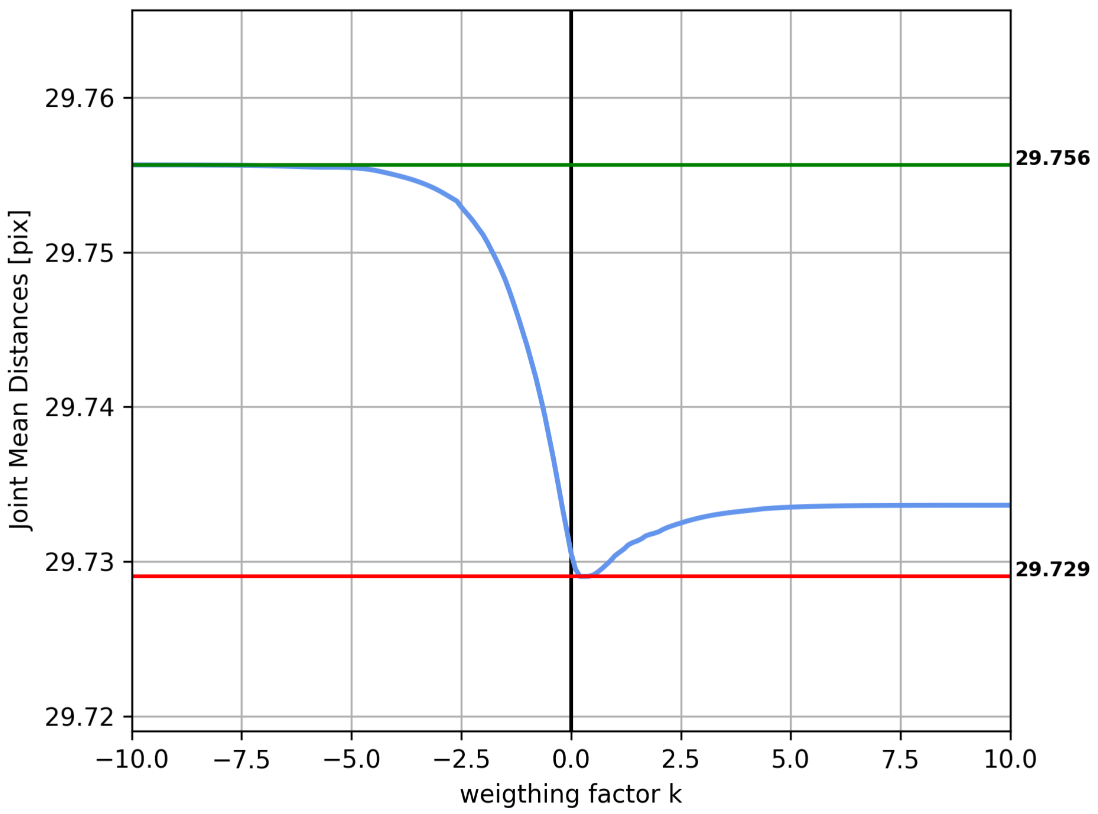


Figure 5.10: Impact of the weighting parameters to the tracking predictions. The curve shows the mean difference of the predicted joints in pixel, evaluated at different weighting factors k with a maximal history amount of ten samples.

The data in Figure 5.10 show a small peak of a lowest distance error that is located at $k = 0.3$. In order to maximize the tracking accuracy that value is used for the here presented system. The final resulting weighting factors for specific history sample that is hold available in the tracker are visualized in Figure 5.11

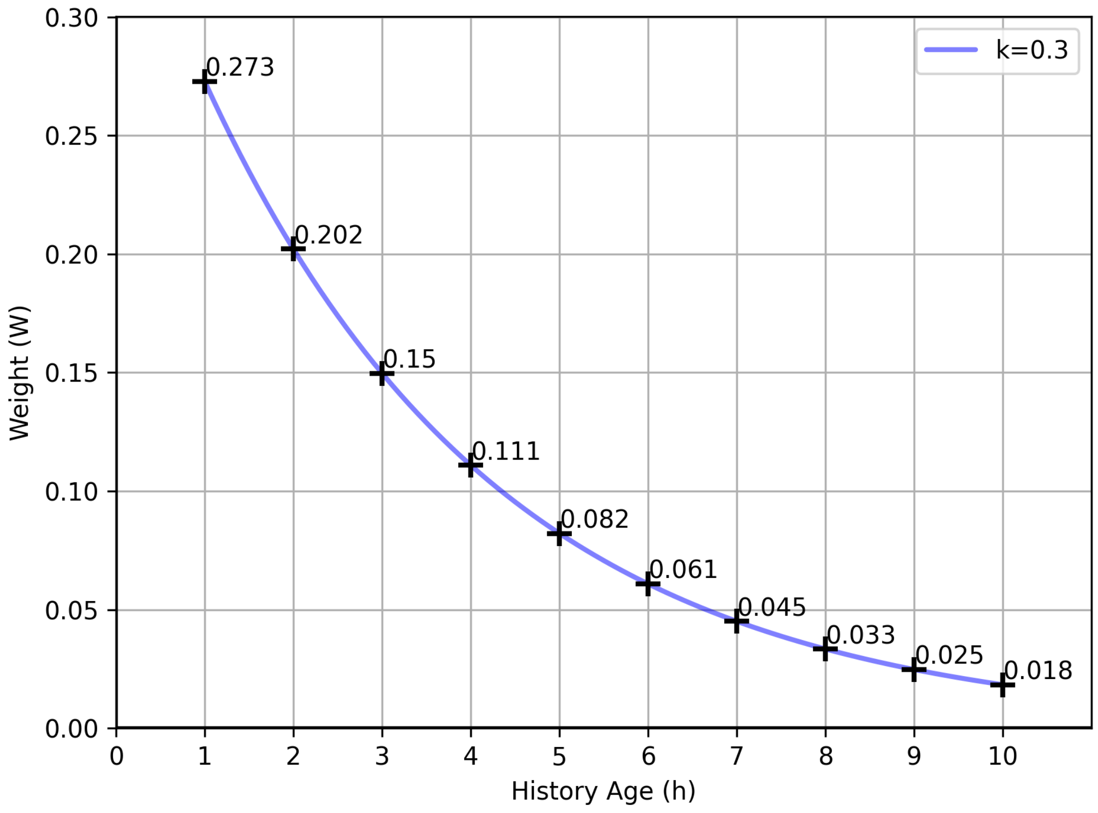


Figure 5.11: The graph shows the specific history weights for $k = 0.3$ used in the tracking system.

The here presented evaluation shows, that the introduced history weighting has almost no impact to the final tracking result. This completely conflicts with the author's hypothesis that a specific weighting should help the tracking system to give more precise pose predictions. The main reason for this result could be, that the recorded data samples only contain strictly linear planed trajectory without common used methods like interpolation to smooth and accelerate trajectories in industrial robotics. It is part of the author's future work to examine if the introduced systematic weighting can support the tracking in a more realistic scenario.

5.6.3 Evaluation of the Tracking System

To avoid an impact of the employed bounding box detector's accuracy on the tracking mechanism evaluation, this main evaluation is based on the ground truth data that is available in the *RoPose* datasets and thus is again performed for all 29.236 samples. The main purpose of the tracking system is to predict the future position of a target's bounding box when the bounding box detector of the main pipeline fails. To evaluate the tracker's accuracy, again the IoU metric was employed to compare the known GT and the predicted bounding box estimated by the tracking system for both, the untouched as well as virtual displaced joint positions. For the ROI prediction, the best performing parameter set, e.g. for the history weighting, detailed at Section 5.6.2 was used. Because this evaluation was based on the given GT of the Datasets, there is no possibility for the tracker to generate false positive predictions and the *Precision* will always be 1.0 (cf. Eq. 2.97 with $FP = 0$). Accordingly Figure 5.12 shows only the development of the F1-scores which are solely affected by the change of the recall.

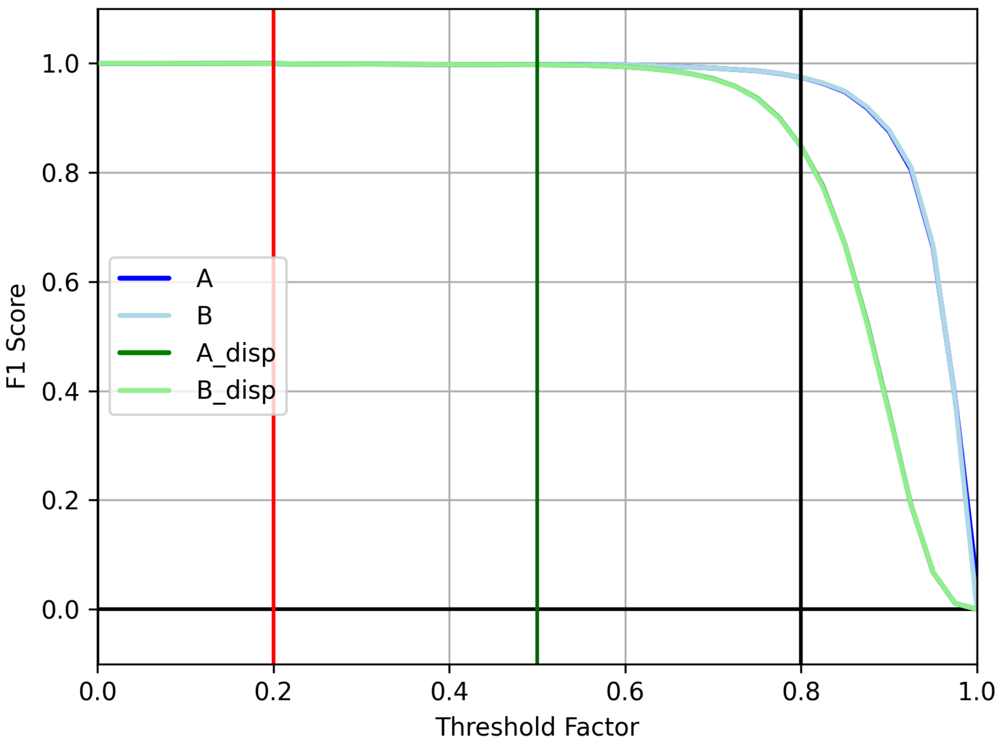


Figure 5.12: F1-score of the tracking mechanism based on GT data. The graph combines the evaluation based on untouched GT values (blue) as well as based on virtually displaced values (green). The Recall-scores are visualized for both methods based on all poses of the kinematic chain (Method A) and only on the corner positions (Method B) of the bounding box (cf. Section 4.2.3). The tracker used a maximum history of ten and a weighting factor of $k = 0.3$. Details of the employed metric are given in Section 2.7.

The *F1-scores* show only marginal differences between both methods. The main reason for this is likely that the bounding box for the tracking is also reconstructed from the pose constellation itself. This shows that the tracking system can also work on the lower parameter dimension of just two BB corners instead of all observed joints of the robot. The data allows the conclusion, that method *A* performs slightly better under more realistic conditions. Method *B* even outperforms *A* when it comes to more ideal conditions and also will need less computational effort because of the mentioned dimension reduction. Therefore it is recommended to choose the bounding box based tracking to stabilize the pre-detector if the marginal performance gain is irrelevant for the application. As the bounding boxes fed into the tracking system are reconstructed based on the extracted poses, the bounding box tracker should just be as robust against occlusion as the pose based tracker. Table 5.7 also proves marginal advantages for *A* when the bare numbers are considered.

Table 5.7: Evaluation key values of the tracker's predictions.

		F1@0.2	**F1@0.5**	**F1@0.8**	**mIoU**
A	Pose	0.999	0.998	0.975	0.928
B	B.Box	0.999	0.998	0.974	0.928
A_disp	Pose	0.999	0.997	0.847	0.836
B_disp	B.Box	0.999	0.997	0.847	0.836

Evaluation key values of the tracker's predictions evaluated for both methods based on all poses of the kinematic chain or only on the corner positions of the bounding box (cf. Section 4.2.3). $F1@$ dedicates to the F1-score evaluation at a specific threshold.

The gathered data show that the tracking system itself shows a higher performance compared to the raw bounding box detector (cf. Table 5.2). This is expected because the above evaluation of the tracker was based on the ideal GT and is not able to predict bounding boxes on a raw image but predicts the current position out of the known history and its main purpose is to support the ROI estimation if it fails.

A first quantification of the real tracking-impact is the number of restored bounding boxes compared to the system with deactivated tracking and the corresponding mean IoUs of the recovered bounding boxes. Table 5.8 shows the final amount of false negative prediction of both systems and both tracking methods. Because the main prediction system is now based on YOLO and thus involved a learning process, the following comparisons are now solely based on the evaluation dataset of *RoPose* which sums up to 3201 examples.

Table 5.8: Performance comparision of the tracking system.

	FN-Count	rec. mIoU
No Tracking	561	-
Tracking A	36	0.636
Tracking B	37	0.644

Performance key values of systems with and without activated tracking. The table shows the amount of false negative predictions of each system and also the mean IoUs of the recovered bounding boxes when tracking is involved.

To also evaluate the positive effect of the tracking system to the final pose estimation where all subsystems are involved, the evaluation from Section 5.4 was performed in the same way but with activated bounding box tracking (Method B) and based on the *YOLO* predicted BB instead of the given GT. The impact on the final PCK results are shown in Figure 5.13.

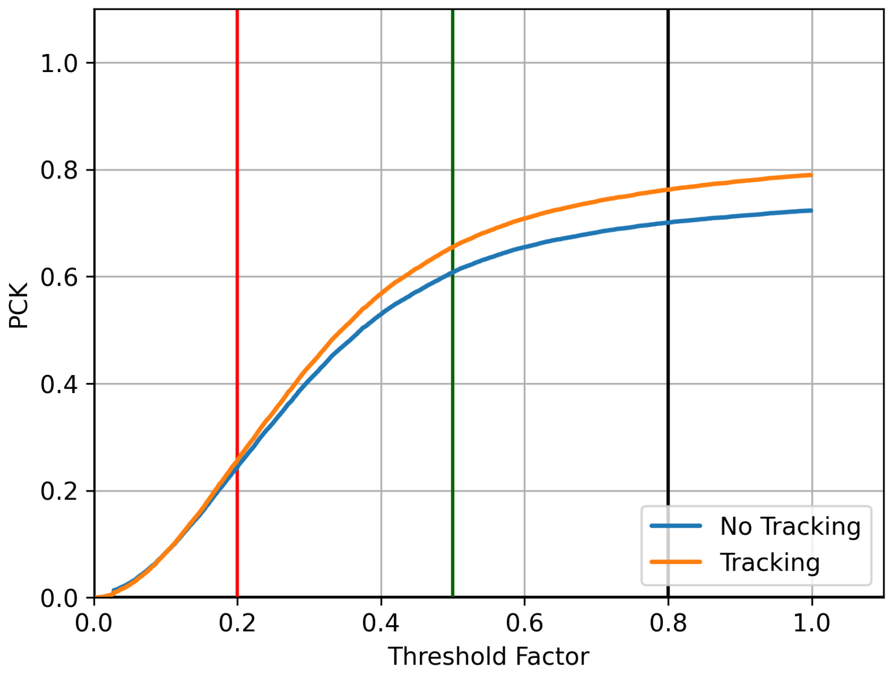


Figure 5.13: Impact of the tracking system on the system's performance based on a PCK comparison of all joints combined. Details of the employed metric are given in Section 5.2.

At this point, it is clear that additional tracking helps to stabilize the pre-detection and improves the overall performance of the system. Thus, an additional tracking solution should always be considered when designing systems similar to *RoPose* .

The basic pose prediction used at the time of the submission of this thesis could maybe significantly improved by implementing other SOTA, mainly machine learning and CNN based [246, 247] or Kalman filter based [248, 249, 250] tracking approaches . As better tracking will lead to higher accuracy of the main system and would also improve the impact of the autonomous fine-tuning pipeline detailed in Section 4.3, it is seen as a future task to improve the tracker's prediction accuracy.

5.7 Self-supervised finetuning

To be able to rate the performance impact of the self-supervised finetuning introduced in Section 4.3 the very same model that was trained for the base evaluation in Section 5.4 and 5.3 was taken as the benchmark and will be referred to as model *A*, respectively the finetuned model as model *B*. The dataset for this finetuning process has been taken from the evaluation datasets and has been reduced to the largest dataset with the same viewport (overall 373 samples). Because the *RoPose* model that is responsible for the keypoint regression stays untouched, it does not make sense to run another evaluation on that performance itself, thus this part will concentrate on the bounding box detector's performance on all datasets used within this finetuning process before and after the finetuning. For the fitting process, the SGD optimizer with a learning rate of $lr = 1e^{-5}$ was used. No *momentum* or other methods were used within the finetuning process.

As this subsystem also produces bounding boxes, again the IoU metric (cf. Section 2.7.2) was employed. Figure 5.14 shows the mean IoU reached by the model before and after some finetuning epochs.

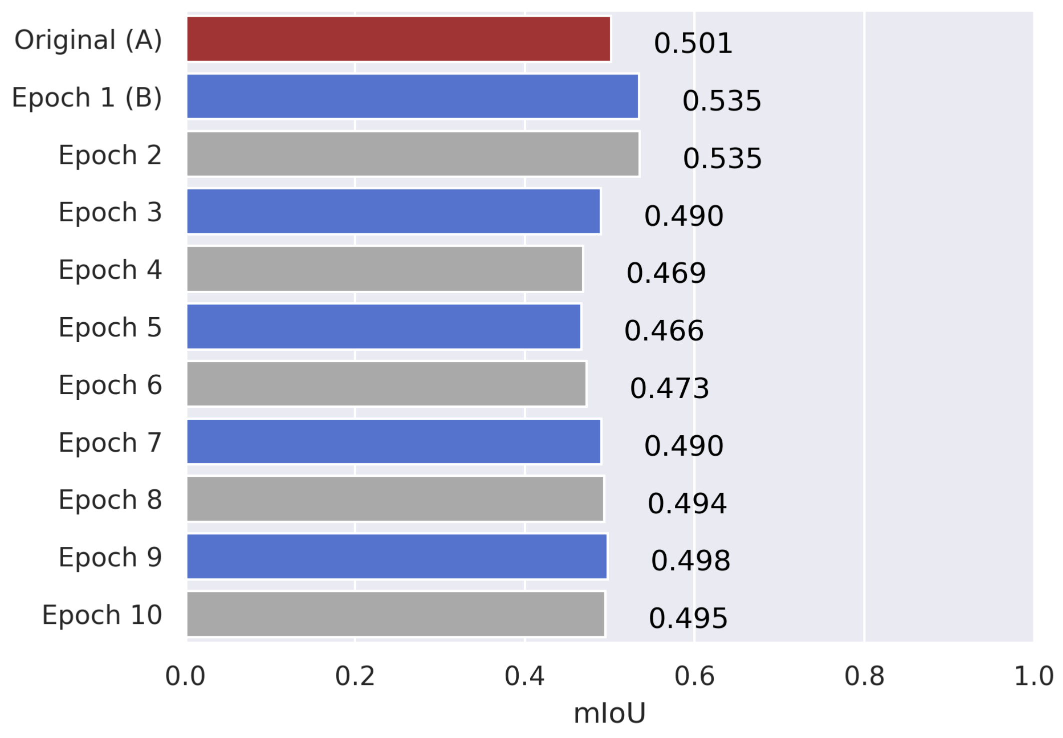


Figure 5.14: Mean IoUs of the finetuned YOLO-based *RoPose* bounding box detector. The numbers need to be compared to the initial trained model's performance (red bar) and show the increase after various finetuning iterations. Details of the employed metric are given in Section 2.7.

The data in Figure 5.14 shows that the self-supervised finetuning approach is able to increase the overall performance of the bounding box detector after the first view epochs. The bars also show the impact of additional finetuning *epochs*, although they are not necessary for a real system's performance comparison. The problem here is that it can not be expected that the same exact image input is available multiple times in a system that is deployed to the real world because of the variance caused by changed light conditions, different human- and robot poses and also changed visual appearances. Regardless, it is interesting to see how the system behaves in such a context and the data shows how fast such a model overfits to the given context and even reduces the overall performance after three epochs. To keep an eye on a realistic impact, this evaluation will concentrate on the results after a single finetuning iteration as shown in Figure 5.15 and also Table 5.9.

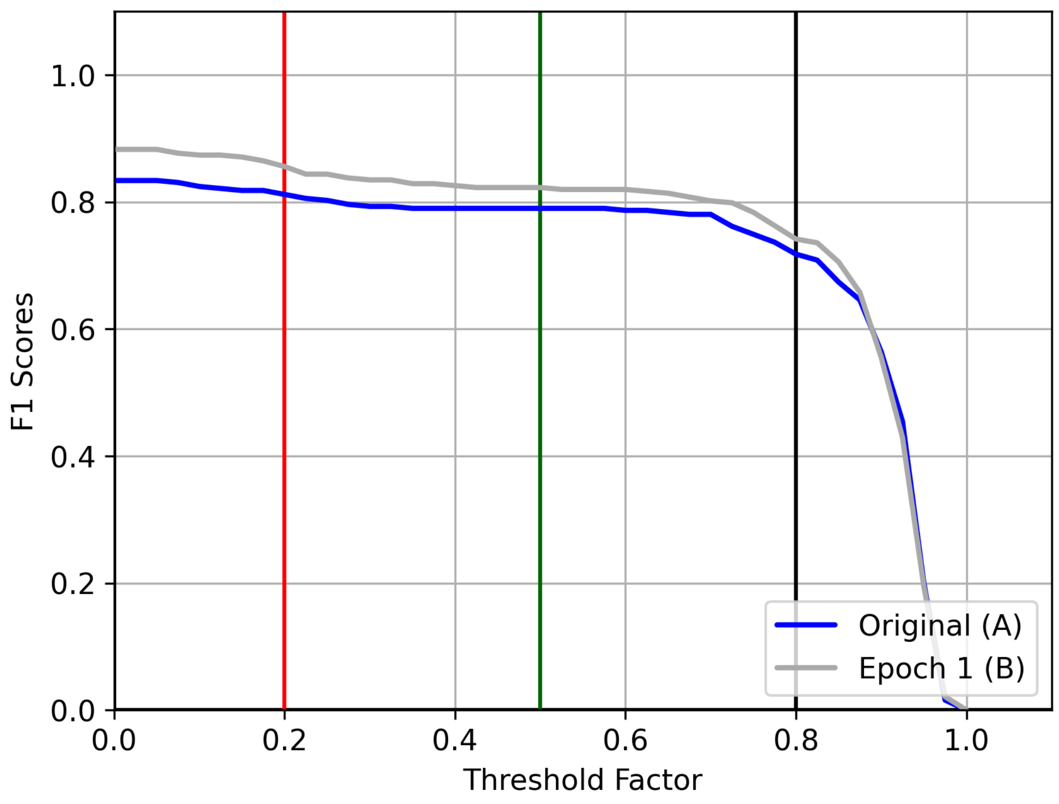


Figure 5.15: F1-scores of the finetuned YOLO-based *RoPose* bounding box detector evaluated at various thresholds th_{bb}. As the origin model's performance is visualized by the blue curve, the gray curve shows the increase after one autonomous finetuning iteration. Details of the employed metric are given in Section 2.7.

Table 5.9: Evaluation key values of the fine-tuned bounding box detector.

Model	AP	F1@0.2	F1@0.5	F1@0.8	mIoU
A	0.701	0.812	0.79	0.718	0.501
B	**0.772**	**0.856**	**0.823**	**0.742**	**0.535**

The performance analysis shows that such a finetuning process is able to adapt the pre-trained model to a given special domain where the system is deployed. Again, the threats that were already discussed in Section 4.3 should always be considered. Such a procedure could lead to a heavy over-fitting of the model to a given domain and could so reduce its global performance which could after all also be seen as the wanted behavior if the estimation accuracy wants to be maxed out at a unique given targeted domain. From a more general point of view, the so generated new training samples should be injected in the available database in a balanced manner. Such datasets could then be used in a general training and finetuning process to also improve the overall performance.

It is clear that the mechanism of finetuning also involves a learning, thus optimization, process and it is certain that things like the chosen optimizer and the learning rate will strongly affect the model's performance and a better tweaked finetuning process could improve the final impact of the finetuning system. This evaluation should only indicate the positive impact of the process to one of the most crucial parts of the processing pipeline. As the finetuning of deep networks can easily be considered as a separate research field and there is some literature available that tackles exactly that topic [239, 251, 252], many methods offer various ways to achieve a performance increases are available. However, the improvement of the overall training process and the adjustment of the involved meta parameters is beyond the scope of the present thesis but is one of the author's future research interest because this process should also have an impact on human pose estimation systems that rely on a pre-detection system.

6 Final Application

The final *RoPose* application is able to extract important information about dynamic collaborative environemnts (cf. 1.2). It combines all different subsystems as the bounding box pre-detection (cf. Section 3.3.5), the CNN-based heatmap regression pipeline for industrial robot arms and humans (cf. Section 3.3.3.2 and 4.1) as well as the tracking and future area prediction system (cf. Section 4.2.3) with the optional self-supervised fine-tuning mechanism (cf. Section 4.3). The abstract diagram in Figure 6.1 should give a complete overview of the involved components.

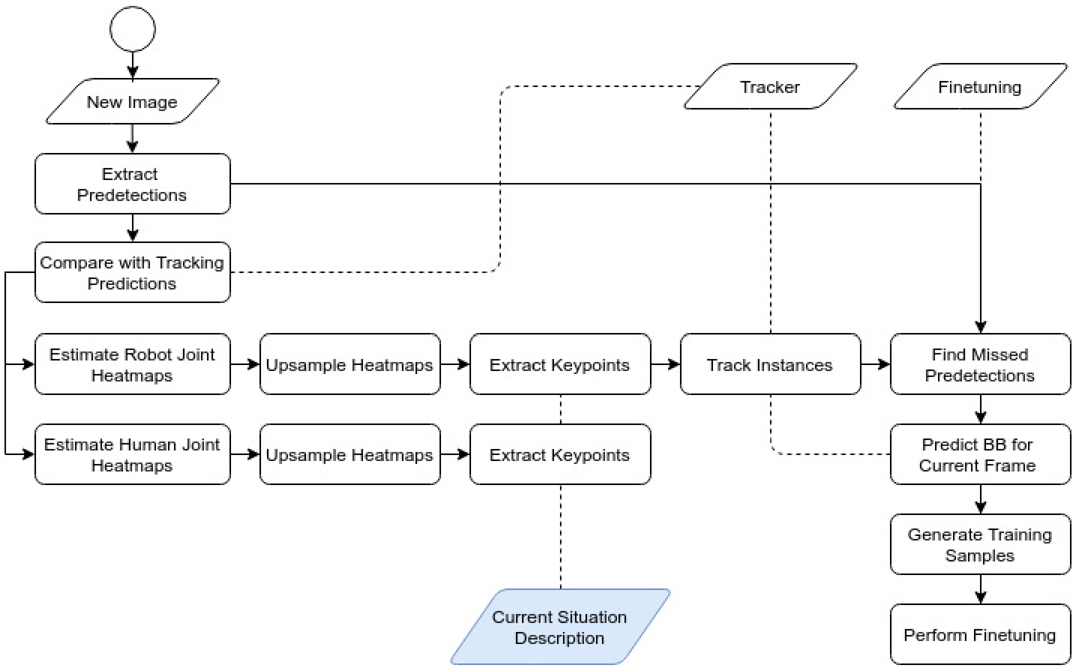

New Image
Tracker
Finetuning
Extract Predetections
Compare with Tracking Predictions
Estimate Robot Joint Heatmaps
Upsample Heatmaps
Extract Keypoints
Track Instances
Find Missed Predetections
Estimate Human Joint Heatmaps
Upsample Heatmaps
Extract Keypoints
Predict BB for Current Frame
Generate Training Samples
Current Situation Description
Perform Finetuning

Figure 6.1: Abstract overview of the proposed final application. After a new image is available the pre-detection system will extract the ROIs for humans and robot objects on the image. These ROIs will be compared and probably concatenated with not detected but already tracked robot instances and will then get preprocessed for both pose regression networks. The batch-wise processed and upsampled probability heatmaps delivered by the two independent CNNs can than be used to extract the 2D keypoints. This information already describes the current image pose situation which can be used in higher-level applications. The given information will also be fed into the tracking system to track the instances and also filter potential FNs examples from the pre-detection to generate finetune training examples that can be used to tweak the pre-detector to the current domain.

Every pose estimation cycle starts with the extraction of the ROIs for all targets. These area estimations will be compared with the potentially available predictions, of the tracking system, which will probably add additional ROIs to the robot targets. The resulting image areas define the input for the separate heatmap regression networks for both target types and need to be preprocessed to fit the network's input constraints. After the propagation through the CNNs, the upsampled probability heatmaps will be used to extract the 2D pose estimation of the target's joints. This joint information can then be used in many other higher-level applications and also forms the input for the proposed tracking system. The tracker will calculate the pose descriptor and compare these results with the already known tracked instances. The given information can be used to extract FNs from the pre-detector which will lead to new, autonomously annotated, training samples that can be used for online finetuning or to build up a growing database for a general training session.

The proposed architecture offers a flexible way to tune and adjust separate parts of the processing pipeline without influencing the other elements. This is primarily considered as an advantage for the *RoPose* parts just as the pre-detector and heatmap regression, because the training database can still be seen as limited and not very diverse compared to the available data sources used for the system parts that target humans. This offers a convenient way for potential third-party users to adapt the system to their unique robot. As a proposed solution should never be stated as perfect, there is always some potential for improvements. On the computer system available[1], the whole pipeline performed at ~ 12 Frames Per Second (FPS). The corresponding processing time of $0,083ms$ is considered as the system's main drawback because this may not be fast enough to reliably track very fast movements of industrial robot systems. Especially when looking at safety aspects, the frame rate needs to be significantly improved to satisfy such application constrains. As Ludl *et al.* presented, it is possible to increase a similar human pose estimation based system by turning off the pre-detection and stick to the tracking prediction after an object was successfully tracked for the first time[6]. Such a modification could also be implemented in the given application and should improve the achievable overall framerate under certain circumstances.

Besides the bare computational performance, there is also a lack of accuracy. As usual, there is nothing like *accurate enough* and if an application is supposed to somehow reach a GT-like prediction accuracy, there is the need for a more accurate GT. However, as the evaluation shows, the main drawback relating to the accuracy is the sometimes large absolute outliers which definitely should be tackled in further applications. One possible solution could be a combination of the here presented simple baseline heatmap regression and the related models from our first publications [I, II] , which will likely result in an increased accuracy but may in turn affect the final frame rate.

[1]Concrete System Specs: CPU: Intel® i7-8700K | GPU: NVIDIA® GeForce GTX® 1080, CUDA® 11.2.2 and cuDNN 8.1.1.3 | OS: Ubuntu 20.04

7 Conclusion and Future Work

7.1 Conclusion

As the main results, benefits as well as drawbacks, problems and issues with the proposed subsystems and the combining final application were discussed in their respective sections, and in the evaluation itself, a less formal and more general conclusion is left at this point.

The present thesis shows a possible solution for a data-driven pose estimation system based on RGB image data, covering the rarely tackled domain of industrial robot-arms. It shows what is necessary to develop, improve, integrate, deploy, and evaluate such an application. It was proven that the presented CNN based approach can be adapted to the given data and can finally be used to extract 2D joint predictions with certain accuracy and also with a solid framerate. The so available information can now be used by various higher level applications where an observing camera system is available and the extracted 2D information of a robot's kinematic chain may be useful. In addition to this standard robot-pose extraction, the introduced extension to simultaneously estimate human joints within the same application leads to a dense, consistent, and therefore rich information flow.

The exploitation of these extracted robot poses by a novel extrinsic calibration procedure was also introduced in Section 3.4. This calibration method can be used to obtain the 3D pose of a monitoring camera device in relation to the robot's intrinsic coordinate frame when the 3D joint information of the robot itself is also available. This proposed procedure allows for a marker-less extrinsic calibration of the camera systems which enables interesting application for various use cases within an production- and collaborative environment especially when the location of the camera is dynamic or unknown.

In order to generate the necessary data basis to train and evaluate *RoPose*, a dynamic data- generation and recording system (c.f. 3.3.2) has been presented. The system can be used to generate synthetic datasets and also to record real datasets of physically present robots. As the synthetic datasets are naturally fully annotated containing all 3D and projected 2D poses of the robot's joints, the real world examples rely on an extrinsic calibration of the used camera in order to provide these labels. In alternative to traditionally, e.g. marker-based calibration methods, a semi-automatic labeling procedure, exploiting the introduced joint-based calibration method has also been detailed within this thesis and the preceded publications. This calibration method just affords the manual pose labeling of a few frames and so represents a convenient way to enrich the available datasets with probably unknown view angles.

An additional pose tracking module, introduced in Section 4.2.2, was developed to increase the systems overall pose estimation accuracy. The presented descriptor is able to fully describe a 2D kinematic chain of a robot system and so offers a way to track a robot's movement by comparing the pose similarities from frame to frame. The tracking information is than be used to provide a target's bounding box prediction if the Yolo-based pre-detector fails to avoid complete false negatives in a frame and so improves the overall performance.

The tracking procedure was also coupled to an autonomous fine-tuning process that helps to increase the pre-detector's performance on-the-fly. This process offers interesting use cases to adapt a system to new domains, or could simply reduce the amount of data needed to bootstrap the detection models.

While developing *RoPose*, various tools, packages, and base applications have emerged out of this long process (see Section *Open Source Contributions* at the very beginning of this thesis). Luckily, the author is able to release and open-source almost everything with some value in the hope to help other research teams to save some effort or at least to increase the future database for *RoPose*[1].

7.2 Future Work

There still is a long way to go to realize vision sensor-based applications that fully consider all safety aspects (cf. Section 1.2), at least in the given context where the pose estimation targets are industrial robot-arms. As one of the main problems, the lack of a more general model, could be eliminated with a growing, more diverse, and balanced data basis, a major accuracy increase would probably need adjustment of all involved subsystems, approaches and training procedures.

As the autonomous extrinsic calibration process (cf. Section 3.4) is considered crucial for safety applications, the improvement of this part is one of the author's main future research interests. Such accuracy improvements would also increase the overall quality

[1]The used Datasets and the final weights for the RoPose networks can be downloaded at `https://www.thomas-gulde.de/ropose/downloads`

of the recorded real world datasets where the semi-automatic labeling procedure will be used and thus would also help to improve the future GT accuracy which should always be a research interest when working with labeled datasets.

The main weakness of the here presented approach can be found in the generated data basis. As this is the most vulnerable part of every data-driven approach, a data basis can never be large, diverse, general, and balanced enough. One of the main problems is the static robot position within every single dataset. After an extrinsic calibration is performed, the origin of the robot's intrinsic coordinate system is not allowed to move to ensure that the projected joint coordinates really represents the GT. This could be solved by an additional hardware system to track the camera or to track the robot's base after a first calibration was performed. Theses additional pose relations could be used to update the extrinsic camera pose relative to the robot's base in every frame, which will lead to more dynamic and realistic datasets. The integration of such a system is also one of the main technical goals in the near future.

The presented tracking mechanism can not be seen as a perfect solution as well. The fact that the descriptor is based on a camera-plane relative baseline to calculate angles and their respective similarities introduces a hard viewport dependency. This makes it impossible to compare estimated 2D joints from multiple camera views. In order to allow for such a multi-camera tracking the extrinsic calibration information could be used to project the extracted joints into a normalized coordinate space and calculate the descriptor's information based on the than available normalized baseline.

The presented dataset generator is also capable of generating simulated samples based on the *GAZEBO* simulation system. One of the main research interests of the author is to put this simulation on the next level. With the rise of *ROS#* [253] there finally is a convenient way to realize a simulation based on the popular game engine *Unity™*, especially because Ludl *et al.* presented a *Unity™*-based simulation framework which could now be used as a new basis to generate *RoPose* datasets as well [5].

It would also drastically increase the benefits of the system if it would be able to estimate the joints in 3D coordinates instead of 2D. As there is some promising research (cf. Section 3.3.3.1) within the field of 3D joint estimation approaches based on 2D image data, modern sensors also provide more or less accurate, dense 3D data, which could be used to inject the additional dimension into the 2D estimations. At the moment, it is impossible to tell which will outperform the other.

Acknowledgements

It is time to spread some of the biggest *thank-yous* in the world. The first one definitely goes to my supervisor Cristóbal Curio and the whole *Cognitive Systems Group* at Reutlingen University for giving me the chance to do research in the area I always loved and somehow got obsessed with. I am sure it was not always easy to deal with my special way to approach things and I appreciate your patience and guidance for the last seven years.

I also had the luck to be placed in the office besides Dennis Burgermeister. I can't think of a better PhD-colleague and I am very happy that am allowed to call you a friend after almost five years of work, travel, discussion, fear, anger and many many laughs, döners and close to a million nigiri. I learned a lot from you and your scientific view, and I believe it will be hard to find a software guy like you anywhere in the world.

My second advisor Andreas Schilling and all his students and collaborators I was allowed to meet in the last years during the presentations you organized - it was always a pleasure to listen to all your great ideas and having pizza-based talks afterwards.

Finally, big thanks go to all former cogsys-team members, especially Vinu and Micheal for helping me restoring all the machines I destroyed while working part time on my thesis, as well as the whole university service- and professor-team for the great time I had in Reutlingen and all the students that I worked with.

Thanks, Dennis Schmid, for just being as you are. It does not matter how much time we are actually spending together but you will always be one of my favorite humans lurking around on this planet.

Uwe and his extra-nice girl Ramona (i still have no idea how you earned such a diamond) - thanks for the timeless talks we have every time that help me to remain down on this planet.

The same thank also goes to Mark, Sabi and the little Lara for the endless fun and all the time we have spent and we hopefully will spent together.

Christopher you will always be one of my favorite Geislingers, the beers we are having together are allways the best - I hope there are many more to follow. Special thanks also goes to your girl Diana who somehow kept you alive for so many years.

Thanks also goes to Timi for just being the friend and person that you are and also your wife Vanessa for taking care that you won't kill yourself in some kind of stupid handicraft work or because you think you are something like an athlete and "playing" football once a year.

I also won't forget my years-long *nebensitzer* Philipp Schweikle, I am not sure if I would have survived our first semesters without you and I am happy that you still reach out to me every once in a while.

Also a big *thank you* goes to my former mentor Rolf who ignited my passion for computer-vision and accompanied me in my first (more or less) professional experiences of arranging bits to bytes as well as all the great people from the BS-Group for introducing me to the world of robotics - i hope there will be an opportunity to work together again at some day.

I want to thank Paul Brooks especially, for all your lovely and warm words, my lower weight wingman Jochen, Alex the blob and all the other gym people and their families for preventing me gaining another twenty kilos and all the hard, exhausting and fun training- and whiskey-sessions we had together - I hope there will be many many more.

Finally, Stephan Chrischtl (aka Dr. Lovegun) - it does not matter how far away you are or where in the whole world you try to hide yourself the next time, I will always find you because I could not imagine to find a better friend in this world anywhere... ever... Xiaofen, please take care of my budy, I am not sure if he is able to survive a single day without you anymore.

I really can't tell where I would be without the daily and endless support of my parents Elli and Gerhard. I am very proud that I am allowed to call you "Mama und Papa". Even if the message may not be that clear every day - you are the best parents on earth and there have been nights I only could fall asleep because I could be certain that you will always be there for me and now also for my little family.

Thanks also goes to the best imaginable sister Jessica, her husband Stefan, and my two cute goddaughters Lotta and Lina for *just being there*, supporting me whenever there is a need for that, and also for taking me just as I am.

Last and most important - My *Schneckle* and our little girl Maya also known as the most beautiful daughter in the world. Without your support and love I would either have never finished this thesis, or would have submitted it two years earlier. There is no way to express in words what a lucky man I am and how lucky I feel every day...

THANK YOU!

List of Figures

List of Tables

Abbreviations and Symbols

Abbreviations

AI	Artificial Intelligence
2D	Two-Dimensional
3D	Three-Dimensional
CV	Computer Vision
NN	Neural Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network
LSTM	Long Short-Term Memory
DoF	Degrees of Freedom
FPS	Frames Per Second
GT	Ground Truth
ROS	Robot Operating System
PnP	Perspective-N-Points
LSL	Lab Streaming Layer
IoT	Internet of Tthings
RNN	Recurrent Nerual Network
ROI	Region of Interest
C-Space	Configuration Space
HRC	Human-Robot Collaboration

HRI	Human-Robot Interaction
LSTM	Long Short-Term Memory
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
ViSP	Visual Servoing Platform
VGG	Visual Geometry Group
ReLU	Rectified Linear Units
GAN	Generative Adversarial Network
AP	Average Precision
mAP	Mean Average Precision
AUC	Area Under Curve
ROC	Receiver Operating Characteristics
IoU	Intersection Over Union
PCK	Probability of Correct Keypoints
TH	Threshold
SNE	Stochastic Neighbor Embedding
BB	Bounding Box
CPU	Central Processing Unit
GPU	Graphics Processing Unit
SGD	Stochastic Gradient Descent
YOLO	You Only Look Once
SOTA	State of the Art
FP	False Positive
FN	False Negative
TP	True Positive
TN	True Negative
CAP	Substantial Credit Assignment Path
RANSAC	Random Sample Consensus

Symbols

$\boldsymbol{T}$	Transformation matrix
$\boldsymbol{R}$	Rotation matrix
$\vec{t}$	Translation vector
$\vec{p}$	Position Vector (3D point)
$\vec{q}$	Quaternion
F_x	Specific frame or coordinate system
W	Robot workspace
G	Group
$\mathfrak{g}$	Lie algebra
$\boldsymbol{\omega}_\times$	skew symmetric matrix usually defining a Lie algebra
$\boldsymbol{I}_X$	Symmetric identity matrix of size $X \times X$
e	*Euler's number*, mathematical constant $e \approx 2.71828$
x_n	Input variable of an artificial neuron
w_n	Weight factor for an explicit input x_n
C_{bias}	Optional bias therm of an artificial neuron
lr	Learning rate used for optimization
f_{th}	Factor for various thresholds

Units

m	meter
pix	pixel

References

[1] T. Gulde, D. Ludl, and C. Curio, "Ropose: Cnn-based 2d pose estimation of industrial robots," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2018, pp. 463–470.

[2] T. Gulde, D. Ludl, J. Andrejtschik, S. Thalji, and C. Curio, "Ropose-real: Real world dataset acquisition for data-driven industrial robot arm pose estimation," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 4389–4395.

[3] T. Gulde, S. Kärcher, and C. Curio, "Vision-based slam navigation for vibrotactile human-centered indoor guidance," in *European Conference on Computer Vision*, Springer, 2016, pp. 343–359.

[4] D. Ludl, T. Gulde, and C. Curio, "Enhancing data-driven algorithms for human pose estimation and action recognition through simulation," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–10, 2020.

[5] D. Ludl, T. Gulde, S. Thalji, and C. Curio, "Using simulation to improve human pose estimation for corner cases," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2018, pp. 3575–3582.

[6] D. Ludl, T. Gulde, and C. Curio, "Simple yet efficient real-time pose-based action recognition," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2019, pp. 581–588.

[7] G. Baulig, T. Gulde, and C. Curio, "Adapting egocentric visual hand pose estimation towards a robot-controlled exoskeleton," in *Computer Vision – ECCV 2018 Workshops*, Springer International Publishing, 2019, ISBN: 978-3-030-11024-6.

[8] M. Essich, D. Ludl, T. Gulde, and C. Curio, "Learning to translate between real world and simulated 3d sensors while transferring task models," in *2019 International Conference on 3D Vision (3DV)*, IEEE, 2019, pp. 681–689.

[9] J. Wang and E. Olson, "AprilTag 2: Efficient and robust fiducial detection," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016.

[10] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2011, pp. 3400–3407.

[11] I. A. Sucan and S. Chitta. "MoveIt!" (), [Online]. Available: `http://moveit.ros.org` (visited on 04/04/2022).

[12] Swartz Center for Computational Neuroscience. "Labstreaminglayer." (2019), [Online]. Available: `https://github.com/sccn/labstreaminglayer` (visited on 04/04/2022).

[13] B. Gassend. "Dynamic reconfigure." (2015), [Online]. Available: `http://wiki.ros.org/dynamic_reconfigure` (visited on 04/04/2022).

[14] E. Musk, *Twitter post*, Sep. 2018. [Online]. Available: `https://twitter.com/elonmusk/status/984882630947753984` (visited on 04/04/2022).

[15] B. Esmaeilian, S. Behdad, and B. Wang, "The evolution and future of manufacturing: A review," *Journal of Manufacturing Systems*, vol. 39, pp. 79–100, 2016.

[16] A. Rose, B. M. Deros, M. A. Rahman, and N. Nordin, "Lean manufacturing best practices in smes," in *Proceedings of the 2011 International Conference on Industrial Engineering and Operations Management*, vol. 2, 2011, pp. 872–877.

[17] R. Shah and P. T. Ward, "Lean manufacturing: Context, practice bundles, and performance," *Journal of operations management*, vol. 21, no. 2, pp. 129–149, 2003.

[18] P. Ward and H. Zhou, "Impact of information technology integration and lean/just-in-time practices on lead-time performance," *Decision Sciences*, vol. 37, no. 2, pp. 177–203, 2006.

[19] D. Pollard, S. Chuo, and B. Lee, "Strategies for mass customization," *Journal of Business & Economics Research (JBER)*, vol. 14, no. 3, pp. 101–110, 2016.

[20] R. Hämäläinen, M. Lanz, and K. T. Koskinen, "Collaborative systems and environments for future working life: Towards the integration of workers, systems and manufacturing environments," in *The impact of digitalization in the workplace*, Springer, 2018, pp. 25–38.

[21] B. Matthias, S. Kock, H. Jerregard, M. Kallman, I. Lundberg, and R. Mellander, "Safety of collaborative industrial robots: Certification possibilities for a collaborative assembly robot concept," in *2011 IEEE International Symposium on Assembly and Manufacturing (ISAM)*, Ieee, 2011, pp. 1–6.

[22] Reutlingen University. "Kollaborativer routenzug 4.0 (kollro 4.0)." (2016), [Online]. Available: `https://www.esb-business-school.de/forschung/wertschoepfungs-und-logistiksysteme/forschungsprojekte/kollro-40/` (visited on 04/04/2022).

[23] J. Schuhmacher and V. Hummel, "Self-organization of changeable intralogistics systems at the esb logistics learning factory," *Procedia Manufacturing*, vol. 31, pp. 194–199, 2019.

[24] J. M. Alfred Siewe-Reinke. “Konsens-nhe bw-neurorobotik.” (2017), [Online]. Available: `https://www.inf.reutlingen-university.de/forschung/projekte/konsens-nhe/` (visited on 04/04/2022).

[25] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, “Deep learning for computer vision: A brief review,” *Computational intelligence and neuroscience*, vol. 2018, 2018.

[26] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[27] J. Fryman and B. Matthias, “Safety of industrial robots: From conventional to collaborative applications,” in *ROBOTIK 2012; 7th German Conference on Robotics*, VDE, 2012, pp. 1–5.

[28] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding, and B. Matthias, “Safety in human-robot collaborative manufacturing environments: Metrics and control,” *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 882–893, 2015.

[29] P. A. Hancock, D. R. Billings, K. E. Schaefer, J. Y. Chen, E. J. De Visser, and R. Parasuraman, “A meta-analysis of factors affecting trust in human-robot interaction,” *Human factors*, vol. 53, no. 5, pp. 517–527, 2011.

[30] A. D. Dragan, S. Bauman, J. Forlizzi, and S. S. Srinivasa, “Effects of robot motion on human-robot collaboration,” in *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, ACM, 2015, pp. 51–58.

[31] C. Byner, B. Matthias, and H. Ding, “Dynamic speed and separation monitoring for collaborative robot applications–concepts and performance,” *Robotics and Computer-Integrated Manufacturing*, vol. 58, pp. 239–252, 2019.

[32] G. Michalos, S. Makris, P. Tsarouchi, T. Guasch, D. Kontovrakis, and G. Chryssolouris, “Design considerations for safe human-robot collaborative workplaces,” *Procedia CIrP*, vol. 37, pp. 248–253, 2015.

[33] H. A. Pierson and M. S. Gashler, “Deep learning in robotics: A review of recent research,” *Advanced Robotics*, vol. 31, no. 16, pp. 821–835, 2017.

[34] L. Deng and D. Yu, “Deep learning: Methods and applications,” *Foundations and trends in signal processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[35] C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle, “Deep learning for computational biology,” *Molecular systems biology*, vol. 12, no. 7, 2016.

[36] P. Baldi, P. Sadowski, and D. Whiteson, “Searching for exotic particles in high-energy physics with deep learning,” *Nature communications*, vol. 5, p. 4308, 2014.

[37] Q. T. Ain, M. Ali, A. Riaz, A. Noureen, M. Kamran, B. Hayat, and A. Rehman, "Sentiment analysis using deep learning techniques: A review," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 6, 2017.

[38] P. P. Van Der Smagt and B. J. Krose, "A real-time learning neural robot controller," in *Proceedings of the 1991 International Conference on Artificial Neural Networks*, vol. 1, 1991, pp. 351–356.

[39] F. L. Lewis, K. Liu, and A. Yesildirek, "Neural net robot controller with guaranteed tracking performance," *IEEE Transactions on Neural Networks*, vol. 6, no. 3, pp. 703–715, 1995.

[40] F. L. Lewis, A. Yesildirek, and K. Liu, "Multilayer neural-net robot controller with guaranteed tracking performance," *IEEE Transactions on Neural Networks*, vol. 7, no. 2, pp. 388–399, 1996.

[41] R. Köker, "A genetic algorithm approach to a neural-network-based inverse kinematics solution of robotic manipulators based on error minimization," *Information Sciences*, vol. 222, pp. 528–543, 2013.

[42] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with large-scale data collection," in *International symposium on experimental robotics*, Springer, 2016, pp. 173–184.

[43] A. A. Apolinarska, M. Pacher, H. Li, N. Cote, R. Pastrana, F. Gramazio, and M. Kohler, "Robotic assembly of timber joints using reinforcement learning," *Automation in Construction*, vol. 125, p. 103 569, 2021.

[44] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 6023–6029.

[45] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, A. M. Agogino, A. Tamar, and P. Abbeel, "Reinforcement learning on variable impedance controller for high-precision robotic assembly," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 3080–3087.

[46] D. Morrison, P. Corke, and J. Leitner, "Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach," *Robotics: Science and Systems XIV*, pp. 1–10, 2018.

[47] X. Xiao, B. Liu, and P. Stone, "Agile robot navigation through hallucinated learning and sober deployment," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 7316–7322.

[48] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2017, pp. 3357–3364.

[49] D. K. Kim and T. Chen, "Deep neural network for real-time autonomous indoor navigation," *arXiv preprint arXiv:1511.04668*, 2015.

[50] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 2371–2378.

[51] M. Z. Alom, M. Hasan, C. Yakopcic, T. M. Taha, and V. K. Asari, "Inception recurrent convolutional neural network for object recognition," *Machine Vision and Applications*, vol. 32, no. 1, pp. 1–14, 2021.

[52] K. Li, G. Wan, G. Cheng, L. Meng, and J. Han, "Object detection in optical remote sensing images: A survey and a new benchmark," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 159, pp. 296–307, 2020.

[53] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *arXiv preprint arXiv:1905.05055*, 2019.

[54] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[55] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[56] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[57] A. Milioto, P. Lottes, and C. Stachniss, "Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in cnns," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 2229–2235.

[58] B. De Brabandere, D. Neven, and L. Van Gool, "Semantic instance segmentation for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 7–9.

[59] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[60] F. Liu, G. Lin, and C. Shen, “Crf learning with cnn features for image segmentation,” *Pattern Recognition*, vol. 48, no. 10, pp. 2983–2992, 2015.

[61] G. Li and Y. Yu, “Visual saliency based on multiscale deep features,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5455–5463.

[62] X. Liu, H. Zhao, M. Tian, L. Sheng, J. Shao, S. Yi, J. Yan, and X. Wang, “Hydraplus-net: Attentive deep features for pedestrian analysis,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 350–359.

[63] Z. Luo, A. Mishra, A. Achkar, J. Eichel, S. Li, and P.-M. Jodoin, “Non-local deep features for salient object detection,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017.

[64] J. Ma, X. Jiang, A. Fan, J. Jiang, and J. Yan, “Image matching from handcrafted to deep features: A survey,” *International Journal of Computer Vision*, vol. 129, no. 1, pp. 23–79, 2021.

[65] W. Chu and D. Cai, “Deep feature based contextual model for object detection,” *Neurocomputing*, vol. 275, pp. 1035–1042, 2018.

[66] T. Kong, F. Sun, C. Tan, H. Liu, and W. Huang, “Deep feature pyramid reconfiguration for object detection,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 169–185.

[67] O. Özyeşil, V. Voroninski, R. Basri, and A. Singer, “A survey of structure from motion*.,” *Acta Numerica*, vol. 26, pp. 305–364, 2017.

[68] A. R. Widya, A. Torii, and M. Okutomi, “Structure from motion using dense cnn features with keypoint relocalization,” *IPSJ Transactions on Computer Vision and Applications*, vol. 10, no. 1, pp. 1–7, 2018.

[69] J. Wang, Y. Zhong, Y. Dai, S. Birchfield, K. Zhang, N. Smolyanskiy, and H. Li, “Deep two-view structure-from-motion revisited,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 8953–8962.

[70] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, “Flownet: Learning optical flow with convolutional networks,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2758–2766.

[71] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, “Flownet 2.0: Evolution of optical flow estimation with deep networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2462–2470.

[72] R. de Queiroz Mendes, E. G. Ribeiro, N. dos Santos Rosa, and V. Grassi Jr, “On deep learning techniques to boost monocular depth estimation for autonomous navigation,” *Robotics and Autonomous Systems*, vol. 136, p. 103 701, 2021.

[73] L. Madhuanand, F. Nex, and M. Y. Yang, “Self-supervised monocular depth estimation from oblique uav videos,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 176, pp. 1–14, 2021.

[74] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, “Deep ordinal regression network for monocular depth estimation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2002–2011.

[75] D. Xu, E. Ricci, W. Ouyang, X. Wang, and N. Sebe, “Multi-scale continuous crfs as sequential deep networks for monocular depth estimation,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017.

[76] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, “Deeper depth prediction with fully convolutional residual networks,” in *2016 Fourth international conference on 3D vision (3DV)*, IEEE, 2016, pp. 239–248.

[77] F. Liu, C. Shen, and G. Lin, “Deep convolutional neural fields for depth estimation from a single image,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5162–5170.

[78] N. Schneider, F. Piewak, C. Stiller, and U. Franke, “Regnet: Multimodal sensor registration using deep neural networks,” in *2017 IEEE intelligent vehicles symposium (IV)*, IEEE, 2017, pp. 1803–1810.

[79] G. Iyer, R. K. Ram, J. K. Murthy, and K. M. Krishna, “Calibnet: Geometrically supervised extrinsic calibration using 3d spatial transformer networks,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 1110–1117.

[80] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: An open-source robot operating system,” in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[81] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, IEEE, vol. 3, 2004, pp. 2149–2154.

[82] J. Diebel, “Representing attitude: Euler angles, unit quaternions, and rotation vectors,” *Matrix*, vol. 58, no. 15-16, pp. 1–35, 2006.

[83] L. Euler, “Formulae generales pro translatione quacunque corporum rigidorum,” *Novi Commentarii academiae scientiarum Petropolitanae*, pp. 189–207, 1776.

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

[85] E. G. Hemingway and O. M. OŔeilly, “Perspectives on euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments,” *Multibody System Dynamics*, vol. 44, no. 1, pp. 31–56, 2018.

[86] J. S. Dai, “Euler–rodrigues formula variations, quaternion conjugation and intrinsic connections,” *Mechanism and Machine Theory*, vol. 92, pp. 144–152, 2015.

[87] G. S. Chirikjian, *Stochastic Models, Information Theory, and Lie Groups, Volume 2: Analytic Methods and Modern Applications*. Springer Science & Business Media, 2011, vol. 2.

[88] J. Gallier and D. Xu, “Computing exponentials of skew-symmetric matrices and logarithms of orthogonal matrices,” *International Journal of Robotics and Automation*, vol. 18, no. 1, pp. 10–20, 2003.

[89] E. Eade. “Lie groups for 2d and 3d transformations.” (2013), [Online]. Available: `http://ethaneade.com/lie.pdf` (visited on 04/04/2022).

[90] J.-L. Blanco, *A tutorial on SE(3) transformation parameterizations and on-manifold optimization*, 2010. [Online]. Available: `http://ingmec.ual.es/~jlblanco/papers/jlblanco2010geometry3D_techrep.pdf` (visited on 04/04/2022).

[91] M. Pusa and J. Leppänen, “Computing the matrix exponential in burnup calculations,” *Nuclear science and engineering*, vol. 164, no. 2, pp. 140–150, 2010.

[92] J. J. Duistermaat and J. A. Kolk, *Lie groups*. Springer Science & Business Media, 2012.

[93] D. S. Watkins, *Fundamentals of matrix computations*. John Wiley & Sons, 2004, vol. 64.

[94] R. Tsai, “A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses,” *IEEE Journal on Robotics and Automation*, vol. 3, no. 4, pp. 323–344, 1987.

[95] B. Caprile and V. Torre, “Using vanishing points for camera calibration,” *International journal of computer vision*, vol. 4, no. 2, pp. 127–139, 1990.

[96] G. P. Stein, “Lens distortion calibration using point correspondences,” in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, 1997, pp. 602–608.

[97] Z. Tang, R. G. von Gioi, P. Monasse, and J.-M. Morel, “A precision analysis of camera distortion models,” *IEEE Transactions on Image Processing*, vol. 26, no. 6, pp. 2694–2704, 2017.

[98] E. E. Hemayed, “A survey of camera self-calibration,” in *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance, 2003.*, IEEE, 2003, pp. 351–357.

[99] X. X. Lu, “A review of solutions for perspective-n-point problem in camera pose estimation,” in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1087, 2018, p. 052 009.

[100] C. C. Aggarwal and P. S. Yu, “Outlier detection for high dimensional data,” in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, 2001, pp. 37–46.

[101] L. Quan and Z. Lan, “Linear n-point camera pose determination,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 21, no. 8, pp. 774–780, 1999.

[102] F. Vigueras, A. Hernández, and I. Maldonado, “Iterative linear solution of the perspective n-point problem using unbiased statistics,” in *2009 Eighth Mexican International Conference on Artificial Intelligence*, IEEE, 2009, pp. 59–64.

[103] V. Lepetit, F. Moreno-Noguer, and P. Fua, “Epnp: An accurate o (n) solution to the pnp problem,” *International journal of computer vision*, p. 155, 2009.

[104] L. Ferraz, X. Binefa, and F. Moreno-Noguer, “Very fast solution to the pnp problem with algebraic outlier rejection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 501–508.

[105] S. Choi, T. Kim, and W. Yu, “Performance evaluation of ransac family,” *Journal of Computer Vision*, vol. 24, no. 3, pp. 271–300, 1997.

[106] J. A. Hesch and S. I. Roumeliotis, “A direct least-squares (dls) method for pnp,” in *2011 International Conference on Computer Vision*, IEEE, 2011, pp. 383–390.

[107] Y. Zheng, Y. Kuang, S. Sugimoto, K. Astrom, and M. Okutomi, “Revisiting the pnp problem: A fast, general and optimal solution,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 2344–2351.

[108] S. Urban, J. Leitloff, and S. Hinz, “Mlpnp-a real-time maximum likelihood solution to the perspective-n-point problem.,” *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 3, no. 3, 2016.

[109] I. Morgan, U. Jayarathne, A. Rankin, T. M. Peters, and E. C. Chen, “Hand-eye calibration for surgical cameras: A procrustean perspective-n-point solution,” *International journal of computer assisted radiology and surgery*, vol. 12, no. 7, pp. 1141–1149, 2017.

[110] B. K. Horn and B. G. Schunck, “Determining optical flow,” in *Techniques and Applications of Image Understanding*, International Society for Optics and Photonics, vol. 281, 1981, pp. 319–331.

[111] B. K. Horn and B. G. Schunck, “"determining optical flow": A retrospective,” 1993.

[112] S. Mukherjee, “Ai versus md: What happens when diagnosis is automated?” *The New Yorker*, vol. 3, 2017. [Online]. Available: `https://www.newyorker.com/magazine/2017/04/03/ai-versus-md` (visited on 04/04/2022).

[113] B. Krose and P. v. d. Smagt, *An introduction to neural networks*. 2011.

[114] C. E. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation functions: Comparison of trends in practice and research for deep learning,” in *2nd International Conference on Computational Sciences and Technology*, 2021, pp. 124–133.

[115] B. Xu, N. Wang, T. Chen, and M. Li, “Empirical evaluation of rectified activations in convolutional network,” *arXiv preprint arXiv:1505.00853*, 2015.

[116] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” *arXiv preprint arXiv:1710.05941*, 2017.

[117] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, vol. 17, pp. 261–272, 2020.

[118] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural networks*, vol. 61, pp. 85–117, 2015.

[119] I. Goodfellow, Y. Bengio, and A. Courville. “Deep learning.” (2016), [Online]. Available: `http://www.deeplearningbook.org` (visited on 04/04/2022).

[120] L. Tan and J. Jiang, *Fundamentals of analog and digital signal processing*. AuthorHouse, 2007.

[121] M. Sonka, V. Hlavac, and R. Boyle, *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[122] J. Canny, “A computational approach to edge detection,” *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[123] F. Yu and V. Koltun, “Multi-scale context aggregation by dilated convolutions,” *arXiv preprint arXiv:1511.07122*, 2015.

[124] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *arXiv preprint arXiv:1603.07285*, 2016.

[125] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[126] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[127] F. Seide and A. Agarwal, “Cntk: Microsoft’s open-source deep-learning toolkit,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 2135–2135.

[128] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*, PMLR, 2015, pp. 448–456.

[129] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, “Understanding batch normalization,” in *Advances in Neural Information Processing Systems*, 2018, pp. 7694–7705.

[130] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[131] H. Shrivastava, A. Garg, Y. Cao, Y. Zhang, and T. Sainath, “Echo state speech recognition,” in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2021, pp. 5669–5673.

[132] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *2013 IEEE international conference on acoustics, speech and signal processing*, IEEE, 2013, pp. 6645–6649.

[133] W. Du, Y. Wang, and Y. Qiao, “Rpan: An end-to-end recurrent pose-attention network for action recognition in videos,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 3725–3734.

[134] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[135] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” in *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[136] D. Stathakis, “How many hidden layers and nodes?” *International Journal of Remote Sensing*, vol. 30, no. 8, pp. 2133–2147, 2009.

[137] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[138] A. Newell, K. Yang, and J. Deng, “Stacked hourglass networks for human pose estimation,” in *European conference on computer vision*, Springer, 2016, pp. 483–499.

[139] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[140] S. Sun, Z. Cao, H. Zhu, and J. Zhao, “A survey of optimization methods from a machine learning perspective,” *IEEE transactions on cybernetics*, 2019.

[141] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *Proceedings of COMPSTAT’2010*, Springer, 2010, pp. 177–186.

[142] R. Hecht-Nielsen, “Theory of the backpropagation neural network,” in *Neural networks for perception*, Elsevier, 1992, pp. 65–93.

[143] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient backprop,” in *Neural networks: Tricks of the trade*, Springer, 2012, pp. 9–48.

[144] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

[145] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro, “Image inpainting for irregular holes using partial convolutions,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 85–100.

[146] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[147] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[148] G. Hinton, N. Srivastava, and K. Swersky, “Neural networks for machine learning lecture 6a overview of mini-batch gradient descent,” 2012.

[149] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, “The marginal value of adaptive gradient methods in machine learning,” in *Advances in Neural Information Processing Systems*, 2017, pp. 4148–4158.

[150] D. L. Olson and D. Delen, *Advanced data mining techniques*. Springer Science & Business Media, 2008.

[151] Z. C. Lipton, C. Elkan, and B. Narayanaswamy, “Thresholding classifiers to maximize f1 score,” *arXiv preprint arXiv:1402.1892*, 2014.

[152] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, “Generalized intersection over union: A metric and a loss for bounding box regression,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 658–666.

[153] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, “Convolutional pose machines,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4724–4732.

[154] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, “Realtime multi-person 2d pose estimation using part affinity fields,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[155] Y. Yang and D. Ramanan, “Articulated human detection with flexible mixtures of parts,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 12, pp. 2878–2890, 2012.

[156] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, “2d human pose estimation: New benchmark and state of the art analysis,” in *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, 2014, pp. 3686–3693.

[157] B. Espiau, F. Chaumette, and P. Rives, “A new approach to visual servoing in robotics,” *ieee Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 313–326, 1992.

[158] J. A. Piepmeier, G. V. McMurray, and H. Lipkin, “Uncalibrated dynamic visual servoing,” *IEEE Transactions on Robotics and Automation*, vol. 20, no. 1, pp. 143–147, 2004.

[159] F. Sadeghi, A. Toshev, E. Jang, and S. Levine, “Sim2real viewpoint invariant visual servoing by recurrent control,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4691–4699.

[160] C. Lenz, “Context-aware human-robot collaboration as a basis for future cognitive factories,” Ph.D. dissertation, Technische Universität München, 2011.

[161] C. Lenz and A. Knoll, “Mechanisms and capabilities for human robot collaboration,” in *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, IEEE, 2014, pp. 666–671.

[162] C. Morato, K. N. Kaipa, B. Zhao, and S. K. Gupta, “Toward safe human robot collaboration by using multiple kinects based real-time human tracking,” *Journal of Computing and Information Science in Engineering*, vol. 14, no. 1, p. 011 006, 2014.

[163] M. Kalaitzakis, B. Cain, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios, “Fiducial markers for pose estimation,” *Journal of Intelligent & Robotic Systems*, vol. 101, no. 4, pp. 1–26, 2021.

[164] M. Faessler, E. Mueggler, K. Schwabe, and D. Scaramuzza, “A monocular pose estimation system based on infrared leds,” in *IEEE International Conference on Robotics and Automation (ICRA), 2014*, IEEE, 2014, pp. 907–913.

[165] K. M. Lundeen, S. Dong, N. Fredricks, M. Akula, J. Seo, and V. R. Kamat, “Optical marker-based end effector pose estimation for articulated excavators,” *Automation in Construction*, vol. 65, pp. 51–64, 2016.

[166] J. Bohg, J. Romero, A. Herzog, and S. Schaal, “Robot arm pose estimation through pixel-wise part classification,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 3143–3150.

[167] F. Widmaier, D. Kappler, S. Schaal, and J. Bohg, “Robot arm pose estimation by pixel-wise regression of joint angles,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 616–623.

[168] V. Patil, W. Van Gansbeke, D. Dai, and L. Van Gool, “Dont forget the past: Recurrent depth estimation from monocular video,” *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6813–6820, 2020.

[169] K. Han and K. Hong, “Geometric and texture cue based depth-map estimation for 2d to 3d image conversion,” in *2011 IEEE International Conference on Consumer Electronics (ICCE)*, IEEE, 2011, pp. 651–652.

[170] X. Gratal, J. Romero, and D. Kragic, “Virtual visual servoing for real-time robot pose estimation,” *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 9017–9022, 2011.

[171] H. Lei, F. Zhou, and C. Zhuang, “Multi-stage 3d pose estimation method of robot arm based on rgb image,” in *2021 7th International Conference on Control, Automation and Robotics (ICCAR)*, IEEE, 2021, pp. 84–88.

[172] J. Mišeikis, I. Brijačak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen, “Two-stage transfer learning for heterogeneous robot detection and 3d joint position estimation in a 2d camera image using cnn,” in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 8883–8889.

[173] J. Miseikis, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen, “Multi-objective convolutional neural networks for robot localisation and 3d position estimation in 2d camera images,” in *2018 15th International Conference on Ubiquitous Robots (UR)*, IEEE, 2018, pp. 597–603.

[174] F. Zhou, Z. Chi, C. Zhuang, and H. Ding, “3d pose estimation of robot arm with rgb images based on deep learning,” in *International Conference on Intelligent Robotics and Applications*, Springer, 2019, pp. 541–553.

[175] C. Heindl, S. Zambal, T. Ponitz, A. Pichler, and J. Scharinger, “3d robot pose estimation from 2d images,” *arXiv preprint arXiv:1902.04987*, 2019.

[176] C. Heindl, S. Zambal, and J. Scharinger, “Learning to predict robot keypoints using artificially generated images,” in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2019, pp. 1536–1539.

[177] T. E. Lee, J. Tremblay, T. To, J. Cheng, T. Mosier, O. Kroemer, D. Fox, and S. Birchfield, “Camera-to-robot pose estimation from a single image,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 9426–9432.

[178] B. Xiao, H. Wu, and Y. Wei, “Simple baselines for human pose estimation and tracking,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 466–481.

[179] C. Beleites, U. Neugebauer, T. Bocklitz, C. Krafft, and J. Popp, “Sample size planning for classification models,” *Analytica chimica acta*, vol. 760, pp. 25–33, 2013.

[180] J. Cho, K. Lee, E. Shin, G. Choy, and S. Do, “How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?” *arXiv preprint arXiv:1511.06348*, 2015.

[181] I. Balki, A. Amirabadi, J. Levman, A. L. Martel, Z. Emersic, B. Meden, A. Garcia-Pedrero, S. C. Ramirez, D. Kong, A. R. Moody, *et al.*, “Sample-size determination methodologies for machine learning in medical imaging research: A systematic review,” *Canadian Association of Radiologists Journal*, 2019.

[182] Open Source Robotics Foundation. “Ros geometry package.” (2009), [Online]. Available: `https://github.com/ros/geometry` (visited on 04/04/2022).

[183] E. Marchand, F. Spindler, and F. Chaumette, “Visp for visual servoing: A generic software platform with a wide class of robot control skills,” *IEEE Robotics and Automation Magazine*, vol. 12, no. 4, pp. 40–52, Dec. 2005.

[184] S. Edwards and C. Lewis, “Ros-industrial: Applying the robot operating system (ros) to industrial applications,” in *IEEE Int. Conference on Robotics and Automation, ECHORD Workshop*, 2012.

[185] “Ros-industrial.” (2016), [Online]. Available: `https://rosindustrial.org/` (visited on 04/04/2022).

[186] Open Source Robotics Foundation. “Gazebo robot simulation made easy.” (2014), [Online]. Available: `http://gazebosim.org/` (visited on 04/04/2022).

[187] UNIVERSAL ROBOTS. “Universal robots.” (2019), [Online]. Available: `https://www.universal-robots.com/` (visited on 04/04/2022).

[188] F. Messmer, K. Hawkins, S. Edwards, S. Glaser, and W. Meeussen. “Universal_robot.” (2019), [Online]. Available: `https://github.com/ros-industrial/universal_robot` (visited on 04/04/2022).

[189] G. Guennebaud, B. Jacob, *et al.* “Eigen v3.” (2010), [Online]. Available: `http://eigen.tuxfamily.org` (visited on 04/04/2022).

[190] C.-C. Wang, “Extrinsic calibration of a vision sensor mounted on a robot,” *ieee Transactions on Robotics and Automation*, vol. 8, no. 2, pp. 161–175, 1992.

[191] K. Daniilidis and E. Bayro-Corrochano, “The dual quaternion approach to hand-eye calibration,” in *proceedings of 13th International Conference on Pattern Recognition*, IEEE, vol. 1, 1996, pp. 318–322.

[192] Z. Zhang, “A flexible new technique for camera calibration,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, 2000.

[193] G. Bradski, “The opencv library.,” *Dr. Dobb’s Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, pp. 120–123, 2000.

[194] D. Tang, T. Hu, L. Shen, Z. Ma, and C. Pan, “Apriltag array-aided extrinsic calibration of camera–laser multi-sensor system,” *Robotics and biomimetics*, vol. 3, no. 1, p. 13, 2016.

[195] E. Westman and M. Kaess, “Underwater apriltag slam and calibration for high precision robot localization,” in *Tech. Rep*, Carnegie Mellon University, 2018.

[196] C. Nissler and Z.-C. Marton, “Robot-to-camera calibration: A generic approach using 6d detections,” in *2017 First IEEE International Conference on Robotic Computing (IRC)*, IEEE, 2017, pp. 299–302.

[197] F. Zhao, T. Tamaki, T. Kurita, B. Raytchev, and K. Kaneda, “Marker based simple non-overlapping camera calibration,” in *2016 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2016, pp. 1180–1184.

[198] Intel Corporation. “Intel® realsense™ technologie.” (2019), [Online]. Available: `https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html` (visited on 04/04/2022).

[199] Intel Corporation. “Ros wrapper for intel® realsense™ devices.” (2019), [Online]. Available: `https://github.com/IntelRealSense/realsense-ros` (visited on 04/04/2022).

[200] Microsoft. “Kinect for windows.” (2019), [Online]. Available: `https://docs.microsoft.com/en-us/windows/apps/design/devices/kinect-for-windows` (visited on 04/04/2022).

[201] Y. Chen, Y. Tian, and M. He, “Monocular human pose estimation: A survey of deep learning-based methods,” *Computer Vision and Image Understanding*, vol. 192, p. 102 897, 2020.

[202] Q. Dang, J. Yin, B. Wang, and W. Zheng, “Deep learning based 2d human pose estimation: A survey,” *Tsinghua Science and Technology*, vol. 24, no. 6, pp. 663–676, 2019.

[203] Z. Liu, J. Zhu, J. Bu, and C. Chen, “A survey of human pose estimation: The body parts parsing based methods,” *Journal of Visual Communication and Image Representation*, vol. 32, pp. 10–19, 2015.

[204] R. Poppe, “Vision-based human motion analysis: An overview,” *Computer vision and image understanding*, vol. 108, no. 1-2, pp. 4–18, 2007.

[205] L. Qiu, X. Zhang, Y. Li, G. Li, X. Wu, Z. Xiong, X. Han, and S. Cui, “Peeking into occluded joints: A novel framework for crowd pose estimation,” in *European Conference on Computer Vision*, Springer, 2020, pp. 488–504.

[206] T. Golda, T. Kalb, A. Schumann, and J. Beyerer, “Human pose estimation for real-world crowded scenarios,” in *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, IEEE, 2019, pp. 1–8.

[207] M. Andriluka, U. Iqbal, E. Ensafutdinov, L. Pishchulin, A. Milan, J. Gall, and S. B., “PoseTrack: A benchmark for human pose estimation and tracking,” in *CVPR*, 2018.

[208] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010. [Online]. Available: `http://yann.lecun.com/exdb/mnist/` (visited on 04/04/2022).

[209] M. Defferrard, S. P. Mohanty, S. F. Carroll, and M. Salathé, “Learning to recognize musical genre from audio,” in *WWW ’18 Companion: The 2018 Web Conference Companion*, 2018. [Online]. Available: `https://arxiv.org/abs/1803.05337`.

[210] M. Andriluka, U. Iqbal, E. Ensafutdinov, L. Pishchulin, A. Milan, J. Gall, and S. B. “Posetrack eccv 2018 challenge results.” (2018), [Online]. Available: `https://posetrack.net/workshops/eccv2018/posetrack_eccv_2018_results.html` (visited on 04/04/2022).

[211] M. Defferrard, S. P. Mohanty, S. F. Carroll, and M. Salathé. “Crowdai musical genre recognition.” (2018), [Online]. Available: `https://github.com/crowdAI/crowdai-musical-genre-recognition-starter-kit` (visited on 04/04/2022).

[212] J. Walsh, N. O’ Mahony, S. Campbell, A. Carvalho, L. Krpalkova, G. Velasco-Hernandez, S. Harapanahalli, and D. Riordan, “Deep learning vs. traditional computer vision,” in *Advances in Computer Vision Proceedings of the 2019 Computer Vision Conference (CVC), Volume 1*, Apr. 2019, pp. 128–144.

[213] L. Ke, M.-C. Chang, H. Qi, and S. Lyu, “Multi-scale structure-aware network for human pose estimation,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 713–728.

[214] S.-T. Kim and H. J. Lee, "Lightweight stacked hourglass network for human pose estimation," *Applied Sciences*, vol. 10, no. 18, p. 6497, 2020.

[215] L. Tian, P. Wang, G. Liang, and C. Shen, "An adversarial human pose estimation network injected with graph structure," *Pattern Recognition*, vol. 115, p. 107 863, 2021.

[216] Y. Chen, C. Shen, X.-S. Wei, L. Liu, and J. Yang, "Adversarial posenet: A structure-aware convolutional network for human pose estimation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1212–1221.

[217] C.-J. Chou, J.-T. Chien, and H.-T. Chen, "Self adversarial training for human pose estimation," in *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, IEEE, 2018, pp. 17–30.

[218] W. Yang, W. Ouyang, X. Wang, J. Ren, H. Li, and X. Wang, "3d human pose estimation in the wild by adversarial learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5255–5264.

[219] V. Belagiannis and A. Zisserman, "Recurrent human pose estimation," in *12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017), 2017*, IEEE, 2017, pp. 468–475.

[220] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019.

[221] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, "Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification," *Neurocomputing*, vol. 321, pp. 321–331, 2018.

[222] X. Zhu, Y. Liu, J. Li, T. Wan, and Z. Qin, "Emotion classification with data augmentation using generative adversarial networks," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, 2018, pp. 349–360.

[223] X. Peng, Z. Tang, F. Yang, R. S. Feris, and D. Metaxas, "Jointly optimize data augmentation and network training: Adversarial data augmentation in human pose estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2226–2234.

[224] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint arXiv:1712.04621*, 2017.

[225] J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," in *Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 1*, 2014, pp. 1799–1807.

[226] A. B. Jung, K. Wada, J. Crall, S. Tanaka, J. Graving, C. Reinders, S. Yadav, J. Banerjee, G. Vecsei, A. Kraft, *et al.* "imgaug." (2020), [Online]. Available: `https://github.com/aleju/imgaug` (visited on 04/04/2022).

[227] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[228] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[229] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, Springer, 2016, pp. 21–37.

[230] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[231] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[232] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[233] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[234] C. Ultralytics, *Ultralytics - yolov3 on github*, 2021. [Online]. Available: `https://github.com/ultralytics/yolov3` (visited on 04/04/2022).

[235] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011.

[236] J. Shin, S. Kim, S. Kang, S.-W. Lee, J. Paik, B. Abidi, and M. Abidi, "Optical flow-based real-time object tracking using non-prior training active feature model," *Real-Time Imaging*, vol. 11, no. 3, pp. 204–218, 2005.

[237] S. Bullinger, C. Bodensteiner, and M. Arens, "Instance flow based online multiple object tracking," in *2017 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2017, pp. 785–789.

[238] B. Chu, V. Madhavan, O. Beijbom, J. Hoffman, and T. Darrell, "Best practices for fine-tuning visual classifiers to new domains," in *European conference on computer vision*, Springer, 2016, pp. 435–442.

[239] W. Ouyang, X. Wang, C. Zhang, and X. Yang, “Factors in finetuning deep model for object detection with long-tail distribution,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[240] P. Refaeilzadeh, L. Tang, and H. Liu, “Cross-validation.,” *Encyclopedia of database systems*, vol. 5, pp. 532–538, 2009.

[241] S. Yadav and S. Shukla, “Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification,” in *2016 IEEE 6th International conference on advanced computing (IACC)*, IEEE, 2016, pp. 78–83.

[242] G. E. Hinton and S. Roweis, “Stochastic neighbor embedding,” *Advances in neural information processing systems*, vol. 15, 2002.

[243] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne.,” *Journal of machine learning research*, vol. 9, no. 11, 2008.

[244] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[245] Student, “The probable error of a mean,” *Biometrika*, pp. 1–25, 1908.

[246] A. Doering, U. Iqbal, and J. Gall, “Joint flow: Temporal flow fields for multi person tracking,” *arXiv preprint arXiv:1805.04596*, 2018.

[247] J. Hwang, J. Lee, S. Park, and N. Kwak, “Pose estimator and tracker using temporal flow maps for limbs,” in *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2019, pp. 1–8.

[248] V. Lippiello, B. Siciliano, and L. Villani, “Adaptive extended kalman filtering for visual motion estimation of 3d objects,” *Control Engineering Practice*, vol. 15, no. 1, pp. 123–134, 2007.

[249] S.-K. Weng, C.-M. Kuo, and S.-K. Tu, “Video object tracking using adaptive kalman filter,” *Journal of Visual Communication and Image Representation*, vol. 17, no. 6, pp. 1190–1208, 2006.

[250] P. R. Gunjal, B. R. Gunjal, H. A. Shinde, S. M. Vanam, and S. S. Aher, “Moving object tracking using kalman filter,” in *2018 International Conference On Advances in Communication and Computing Technology (ICACCT)*, IEEE, 2018, pp. 544–547.

[251] D. Zhang, L. Yang, D. Meng, D. Xu, and J. Han, “Spftn: A self-paced fine-tuning network for segmenting objects in weakly labelled videos,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017.

[252] Y.-X. Wang, D. Ramanan, and M. Hebert, “Growing a brain: Fine-tuning by increasing model capacity,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2471–2480.

[253] M. Bischoff. “Ros#.” (2017), [Online]. Available: `https://github.com/siemens/ros-sharp` (visited on 04/04/2022).