Evaluation of the TiAGo Robot for Precision Tasks in PCB Assembly: A Comparative Study

Lukas Flad[1,†], **Mark Leyer**[2], **Robert Hannen**[3,†] **and Chunrong Yuan**[4,†]

[1] *Faculty of Information, Media and Electrical Engineering , University of Sciences Cologne, Germany*
[†] *These authors contributed equally to this work*

This manuscript was compiled on February 10, 2025

Abstract

This study evaluates the feasibility of using the TIAGo robot, a service-oriented manipulator, for high-precision PCB assembly. Through a comparative analysis, placement accuracy, repeatability, and vision-based component recognition are assessed. The study implements feature-based and deep-learning-driven detection techniques, identifying key challenges in component localization. While results indicate that TIAGo possesses potential for flexible automation, limitations in precision and control necessitate further optimization through machine learning and adaptive calibration techniques.

Keywords: *TIAGo robot, Pal-Robotics, PCB assembly, robotic precision, comparative analysis*

Corresponding author: It follows the corresponding author and publisher information. *E-mail address:* lukas.flad@smail.th-koeln.de, mark_uwe.leyer@smail.th-koeln.de, robert_wilhelm.hannen@smail.th-koeln.de
Received: February 9, 2025 **Accepted:** Pending, 2025

Contents

1. Introduction

Automated PCB assembly typically utilizes Cartesian or SCARA robots due to their precision. This study investigates the application of a TIAGo robot, typically used in service environments, in a PCB assembly context to evaluate its potential for industrial tasks.

2. Related Work

Robotic automation in PCB assembly has been widely studied, with SCARA (Selective Compliance Articulated Robot Arm) robots being the predominant choice due to their high precision, speed, and repeatability. The study by Nimon et al. [1] provides a detailed analysis of SCARA robots, such as the DENSO HM-40A04, demonstrate high precision in PCB assembly, with a manufacturer-specified repeatability of $\pm 25\,\mu m$ and an experimentally measured placement deviation averaging $\pm 150.1\,\mu m$ [1].

In addition to traditional industrial robots, vision-based automated inspection has gained prominence in PCB assembly, particularly for quality control and defect detection. Traditional methods, including template matching, edge detection, and statistical classifiers, have been widely used to improve assembly accuracy. Zhou et al. [2] demonstrated that convolutional neural networks (CNNs) enhance component recognition by significantly reducing false negatives in small-scale PCB component detection, outperforming conventional edge-based techniques.

Recent advancements in deep learning-driven object detection have introduced real-time methodologies such as YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector) for PCB inspection. Kim et al. [3] showed that YOLOv5 outperformed classical feature-based detection algorithms, achieving a detection accuracy of 93.5% on industrial PCB datasets. These results indicate that deep learning models trained on custom, high-resolution PCB datasets can significantly enhance the reliability and precision of component recognition.

Despite these advances, service robots such as TIAGo have not been widely adopted for high-precision PCB assembly. Unlike SCARA robots, which operate in fixed workspaces with rigid end-effectors, mobile manipulators like TIAGo introduce challenges in stability, repeatability, and calibration. Studies on service robot applications in industrial automation (e.g., Pal Robotics [4]) emphasize the necessity of adaptive control mechanisms and integrated vision-based positioning to enhance precision in non-structured environments.

3. Methodology

The methodology employed in this research revolves around integrating the TIAGo robot with advanced sensory and control systems to perform precision tasks in PCB assembly. Key components of the methodology include the robotic hardware configuration, software control architecture, vision system setup, and the implementation of kinematic solutions.

3.1. Robotic System Configuration

The core of this experimental setup is the TIAGo robot, equipped with a high-precision camera system and an end-effector suitable for handling electronic components. The robot operates under the control of the Robot Operating System (ROS), which facilitates modular and scalable software architecture.

3.2. Vision System and Camera Calibration

For visual processing, we utilized a mono vision system comprising the head-mounted camera and an additional end-effector camera to enhance the accuracy of component placement. Calibration of these cameras was carried out using OpenCV to ensure precise alignment and measurement capabilities. The calibration process involved determining the intrinsic and extrinsic parameters that describe the camera's geometry and position relative to the robot frame.

3.2.1. Limitations of Stereo Vision in Near-Field Perception

Stereo vision was excluded from this study due to its inherent limitations in near-field depth perception. As established in [5, 6], stereo vision systems rely on disparity computation, which is inherently less reliable at short distances due to the reduced baseline-to-depth ratio. Consequently, depth accuracy diminishes as the observed object approaches the cameras, leading to increased depth uncertainty [7].

Further empirical studies have demonstrated that stereo-based depth estimation exhibits significant error variance in the sub-50 cm range, making it inadequate for high-precision robotic tasks such as component placement in PCB assembly [8, 3]. Moreover, the limited disparity resolution constrains depth differentiation within fine-scale structures, which is critical for precision robotics [9]. In addition, practical implementation constraints within the Robot Operating System (ROS) reinforce the decision to exclude stereo vision. The ROS topic `/xtion/depth_registered/image_raw`, which previously provided structured depth data, is deprecated and no longer available in recent distributions [10]. Similarly, the `sensor_msgs/Image` topic, used for depth data streaming, is unsuitable due to its insufficient layer

granularity in the near field, often yielding no more than five distinguishable depth planes [11].

Given these limitations, this study adopts a monocular camera system. Depth estimation is facilitated through alternative approaches, such as structured light scanning and sensor fusion techniques [12]. These methods compensate for the lack of disparity-based depth computation and provide superior precision in near-field scenarios, as validated in previous robotic applications [13].

3.2.2. Mathematical Model of Camera Calibration

Camera calibration is the process of estimating the parameters of the camera model, which consists of intrinsic and extrinsic parameters. The intrinsic parameters define the camera's internal characteristics, while the extrinsic parameters describe its orientation and position in the world coordinate system.
The standard pinhole camera model represents the relationship between a 3D point $\mathbf{X}_w = (X_w, Y_w, Z_w)^T$ in world coordinates and its corresponding 2D projection $\mathbf{x} = (x, y)^T$ in the image plane:

$$s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K \, [R|t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{1}$$

where:

- s is a scaling factor.
- K is the intrinsic camera matrix:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

  where f_x and f_y are the focal lengths, and (c_x, c_y) is the optical center.
- $[R|t]$ represents the extrinsic matrix, where R is a 3×3 rotation matrix and t is a 3×1 translation vector.

3.2.3. Chessboard-Based Calibration Method

The calibration was performed using a known chessboard (see the following illustration) pattern with dimensions $H \times W$ and square size s. The steps included:

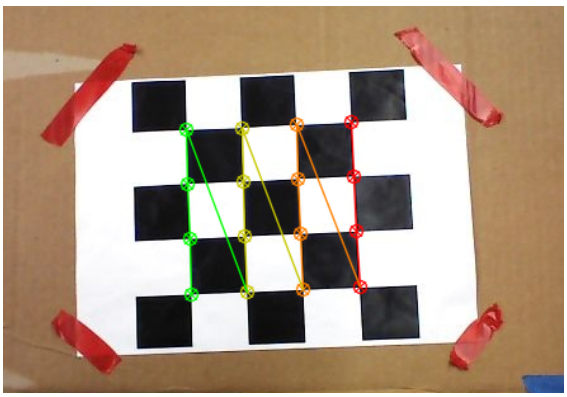

Figure 1. Camera calibration process using a checkerboard pattern to determine intrinsic and extrinsic parameters.

1. Capturing multiple images of the calibration pattern from different angles.
2. Detecting the $H \times W$ internal corners using OpenCV's `findChessboardCorners()`.
3. Refining the detected corners with sub-pixel accuracy using `cornerSubPix()`.
4. Using the detected corners to establish a correspondence between 2D image points and 3D real-world coordinates.
5. Solving for the intrinsic and extrinsic parameters via the Zhang calibration method [2].
6. Computing the distortion coefficients $(k_1, k_2, p_1, p_2, k_3)$ to correct for lens distortions.

The OpenCV calibration function is implemented as follows:

```
ret, mtx, dist, rvecs, tvecs = cv.
    calibrateCamera(objpoints,
    imgpoints, gray.shape[::-1], None,
    None)
```

Code 1. Camera Calibration in OpenCV

where:

- `mtx` is the camera matrix K.
- `dist` contains the distortion coefficients $[k_1, k_2, p_1, p_2, k_3]$.
- `rvecs` and `tvecs` store the rotation and translation vectors for each image.

3.2.4. Distortion Correction

Radial and tangential distortions are modeled using:

$$\begin{aligned} x_{\text{corrected}} &= x\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \\ &\quad + 2p_1 \, xy + p_2\left(r^2 + 2x^2\right) \end{aligned} \tag{3}$$

$$\begin{aligned} y_{\text{corrected}} &= y\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \\ &\quad + p_1\left(r^2 + 2y^2\right) + 2p_2 xy \end{aligned} \tag{4}$$

where $r^2 = x^2 + y^2$ is the squared radial distance from the optical center. The distortion is corrected using OpenCV's `undistort()` function.

```
undistorted_img = cv.undistort(img,
    mtx, dist, None, new_camera_mtx)
```

Code 2. Undistortion in OpenCV

3.2.5. Error Estimation

The accuracy of the calibration is evaluated using the re-projection error:

$$\text{error} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 \tag{5}$$

where $\mathbf{x}_i$ are the detected image points and $\hat{\mathbf{x}}_i$ are the projected points using the estimated parameters.

3.2.6. Experimental Results

After calibration, the obtained camera matrix K was stored in a JSON configuration file for later use. A webcam calibrated with 5 images of a 7x7 checkerboard pattern and achieved a mean re-projection error of $e = 0.013$ pixels, indicating a high level of accuracy.

3.3. Gerber File Processing

To facilitate automated PCB assembly, Gerber files are processed and converted into a machine-readable format. This transformation generates images that accurately represent the PCB layout, serving as a reference for robotic systems to identify precise component placement locations.

3.3.1. Gerber File Parsing

Gerber files contain structured information about the various PCB layers, including traces, pads, holes, outlines, and profiles. The processing pipeline extracts these elements using the `gerber` Python package. The extraction function reads Gerber primitives and categorizes them based on their geometry:

- **Traces:** Representing electrical connections.
- **Pads:** Denoting soldering locations.
- **Holes:** Through-hole component placements.
- **Outlines:** Defining board boundaries.
- **PCB Profile:** Enclosing the entire PCB shape.

These extracted elements are then used to construct a structured representation of the board layout.

3.3.2. Generation of PCB Overlay Image

Once the relevant Gerber elements are extracted, a transparent overlay image is generated using OpenCV. The following color scheme is applied to enhance visibility:

- Green: Outlines of large components.
- Red: Small components and detailed features.
- Blue: PCB pads for accurate placement.
- Magenta: PCB edges and profile.
- Cyan: Detected holes for through-hole components.

Gaussian blurring is applied to smooth the output, ensuring a high-quality overlay.

3.3.3. PCB Recognition and Alignment

To align the PCB overlay with the physical board in a real-world scenario, a camera-based detection system is employed. This system:

1. Captures live footage of the PCB using a camera.
2. Detects key PCB features, such as edges and corners.
3. Computes a homography transformation to align the overlay with the real PCB.
4. Uses iterative averaging over multiple frames to reduce jitter.

This enables real-time visualization of the overlay directly on the PCB surface.

3.3.4. Images of the Process

The following figures illustrate the different stages of the process:

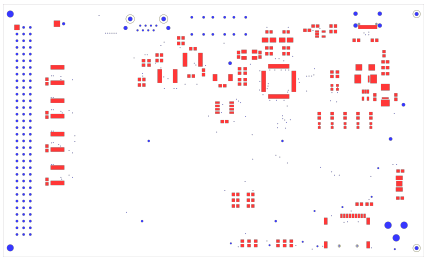

Figure 2. Generated PCB Overlay Image

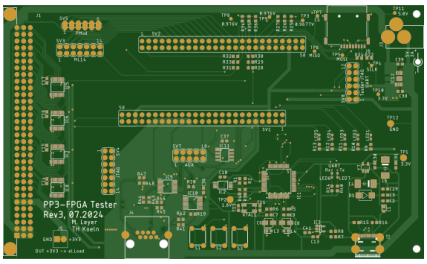

Figure 3. Gerber File Viewed in a PCB Viewer

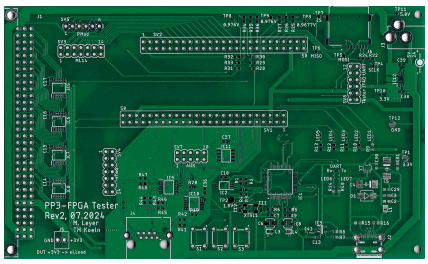

Figure 4. Real PCB Captured via a camera

The combination of Gerber file processing, overlay generation, and real-time PCB recognition enables precise and automated PCB assembly, improving accuracy in component placement, through feature detection.

3.4. Detection Techniques

To ensure the precise identification and placement of components, several detection techniques were used. There methods leverage contour analysis, template matching, and geometric transformation to enhance detection accuracy.

3.4.1. Contour Approximation for Shape Identification

Counter approximation is used to identify and classify geometric shapes of PCBs and PCB components. The approach involves extracting contours from thresholded images and simplifying them using the Douglas-Peucker algorithm [14] (see figure 5)

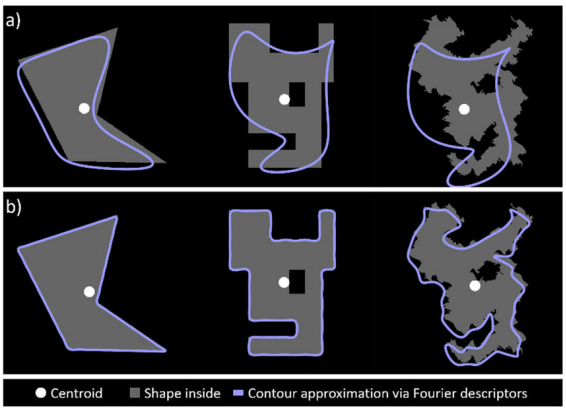

Figure 5. Contour approximation for shape identification [15]

This method uses a parameter to limit the maximum difference between the detected contour and the approximated contour, preventing breaks in the expected straight contour of the PCB. This enables the recognition of distinct component shapes, facilitating their accurate localization on the PCB.

3.4.2. Convex Hull Contour for Large Contour Gaps

Another problem with the contour detection are specular highlights created by bright light sources. The highlights remove parts of the PCB from the thresholded image and create large gaps in the detected contours. To remove those gaps, the convex hull of the contours is used for further processing.

3.4.3. Additional Contour Filtering

To select the right contour belonging to the PCB the contours have to meet additional criteria. Every contour has to have exactly 4 corners to be detected as a possible PCB contour. The remaining contours are filtered by the size of their area to exclude the image border itself, the table the PCB sits on and smaller rectangles on the layout of the PCB itself. A possible additional filtering would only select contour with an average color similar to the average color of the PCB reference image.

3.4.4. Template Matching for Component Recognition

Template matching is employed to detect specific components by comparing predefined reference images with the observed image. This technique measures pixel-wise similarity using methods such as normalized cross-correlation [16]. It is particularly effective for identifying standard PCB elements such as integrated circuits, resistors, and capacitors.

3.4.5. Perspective Transformation for Alignment Correction

To compensate for perspective distortion in the captured images, homography-based transformations are applied. This technique aligns the camera's view with the PCB plane, ensuring that detected component locations correspond accurately to their real-world positions (figure 6.

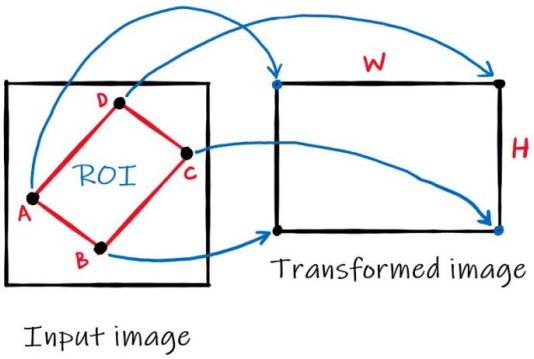

Figure 6. Perspective transformation for alignment correction [17]

The transformation matrix is computed using feature correspondences between the observed image and the reference PCB layout [18]. To validate the perspective projection, the known component locations on the PCB are transformed back onto the image with the inverse perspective projection matrix.

3.4.6. Feature-Based Detection for Component Localization

Feature-based approaches, such as Oriented FAST and Rotated BRIEF (ORB), are employed for the detection of keypoints and descriptors that characterize PCB components, particularly ICs [19]. These features are matched against reference datasets to enhance the robustness of detection, particularly under challenging conditions such as variations in illumination and partial occlusions.

During the evaluation of alternative detection methodologies, a model from Roboflow [20] was considered. However, this approach was ultimately deemed unsuitable for precise component detection due to its limited ability to reliably differentiate individual electronic components. While it demonstrated efficacy in identifying clusters of components on PCBs, it frequently misclassified individual parts, resulting in unreliable outcomes. Rather than correctly distinguishing specific electronic components, the model often detected generic rectangular regions, leading to an increased false-positive rate.

In contrast, ORB-based feature matching exhibited superior performance in extracting distinctive keypoints for IC recognition. Nevertheless, to further enhance detection accuracy and broaden the scope of recognizable components, integrating deep-learning-based object detection models, such as YOLO, could be advantageous. By developing specialized training datasets containing labeled images of individual PCB components, the deep learning capabilities of YOLO could contribute to improved classification accuracy. Recent YOLO implementations, such as those available in the Ultralytics repository [21], offer state-of-the-art object detection functionalities that may be leveraged to achieve higher precision in electronic component identification.

Although preliminary investigations into YOLO-based approaches were conducted, they were not incorporated into the present study. Future research could focus on curating high-quality datasets comprising annotated images of discrete electronic components and training a YOLO model specifically optimized for PCB assembly applications. Such advancements could significantly enhance the reliability and comprehensiveness of automated component recognition systems.

3.4.7. Machine Learning-Based Classification

To enhance detection reliability, machine learning techniques such as convolutional neural networks (CNNs) are integrated into the detection pipeline. The OCR-based component recognition system utilizes EasyOCR [22], an open-source Optical Character Recognition (OCR) library that employs deep learning models for text extraction. EasyOCR is specifically used for identifying alphanumeric labels on integrated circuits (ICs) and other components. By extracting text from PCB and label sheets on which the IC is placed (see figure 7).

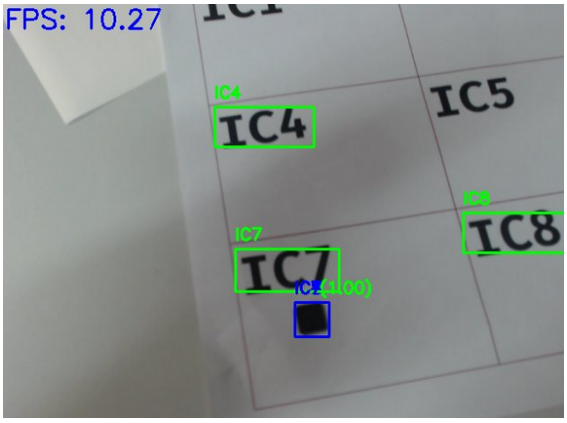

Figure 7. IC part and label detection

4. ROS-Based Control System

The control system was developed using ROS, providing a flexible framework for robot programming. Custom ROS nodes were created to handle different aspects of the robot's operation, from movement control to sensory data processing.

4.1. Introduction

A critical component of this project entails the manipulation of the physical environment. The robot is tasked with the collection of electronic components and subsequent placement and alignment. In alignment with the human work method, it was determined that both components and circuit boards would be situated on a horizontal work surface.

The actual pick-up of electronic components is usually carried out using a vacuum pipette, which is lowered onto the component, and then the vacuum is activated. A component is placed in the same way by placing the component in the intended position in the intended orientation and deactivating the vacuum. As no corresponding hardware was available for this project, picking up and placing a component is only simulated by raising and lowering the torso.

The aforementioned requirements necessitated the movement of the TIAGo manipulator in a horizontal plane, a feat that demanded precise control over three of its seven joints. Subsequent to the initial alignment, it

was imperative that all other joints be maintained in a fixed position to ensure the integrity of the manipulator's operation.

The reflow process employed in the manufacture of printed circuit boards (PCBs) has been demonstrated to compensate for minor deviations in the placement of components. The surface tension of the solder causes slight mechanical movements of components immediately upon the solder's transition to a liquid state in the reflow oven. Based on accumulated experience, it has been determined that, even considering this effect, components must be placed with an accuracy of less than 500 µm.

4.1.1. Software framework on the TIAGo

ROS (Robot Operating System) version 1 has been installed on the TIAGo. Given that the manufacturer does not publish any hardware drivers, control of the TIAGo must be done via ROS.

ROS is a flexible framework for developing robot software, offering a collection of tools, libraries, and conventions that facilitate the creation of complex and robust robotics software. Its modular architecture enables easy integration of hardware and reuse of existing software modules, supporting distributed systems so that different programs and tools can communicate with each other across multiple computers. Initially developed for robot applications, ROS has since been adopted in numerous fields that extend beyond traditional robotics, including the automotive industry and research. ROS uses messages in a publish-subscribe infrastructure to take control of the actuators ans sensors.

4.1.2. Preliminaries

The initial phase of the project entailed an investigation into the precision with which the hardware of the TIAGo can be controlled, given that the torso can only be manipulated through a single position specification. To this end, the height of the torso in meters was configured to be transmitted to the torso_controller via a *JointTrajectoryMessage*. Subsequently, an examination was conducted to ascertain which value ranges resulted in torso movement and which did not. The analysis revealed that values below 0.01 m were not considered and did not result in any movement.

The TIAGo manipulator is controlled via the *arm_controller*, with angles expressed in radians. Similar limitations in resolution have been observed in this context as well. Research has demonstrated that the manipulator motors themselves possess a resolution of 12 bits.[23] Consequently, the angular resolution of the arm is constrained to the following limits:

- *arm_controller* :

$$(360°/2 \cdot \pi) \cdot 0.01 = 0.5732° \tag{6}$$

- Hardware :

$$360°/4096 = 0.0879° \tag{7}$$

To ascertain the minimum movement in meters, the angles in the 45° range are considered, as sin and cos exhibit the most significant changes here. So the minimum step in x- and y-coordinates can be calculated as a minimum angular movement of a single joint:

$$\begin{aligned} l_{\text{arm}} \cdot (sin(45° + r) - sin(45°)) &= 0.44 \cdot 0.0011 \\ &= 0.00048m \end{aligned} \tag{8}$$

The calculations demonstrate that the TIAGo has the capacity to attain the requisite precision in the positioning of the manipulator for the designated project. Nevertheless, given that the pre-installed software lacks the necessary support, it became imperative to devise a customized controller.

4.2. ROS controller

Robot Operating System (ROS) control constitutes the interface for direct control of the hardware drivers. The aforementioned hardware drivers consist, for example, of Proportional-Integral-Derivative (PID) controllers that directly control actuators. Figure 12 in Appendix A.2 presents an overview of the controller data flow in ROS.

4.2.1. Developing a ROS controller

In an initial attempt to develop a controller from the ground up, the researcher sought guidance from available tutorials. However, it was observed that these tutorials proved to be of limited utility. A closer examination revealed that all tutorials were founded upon a singular example from ROS.org. Notably, this example did not involve the development of a controller for actual hardware; rather, it showcased only the fundamental capabilities of the *controller_manager*.

An effort was thus initiated to utilize the existing controller source code to ascertain the necessary structure of a controller and the method by which communication with the hardware is facilitated. A notable challenge in this endeavor is the absence of published source code below the controller level, which precludes the identification of API descriptions. At the hardware level, it was observed that certain PID controllers regulate actuators, such as the joints of the manipulator.

4.2.2. PID controller

A PID (Proportional-Integral-Derivative) controller is a control system that corrects deviations from a setpoint using three components: Proportional, Integral, and Differential components. The proportional component responds to the current deviation by making a directly proportional adjustment. The integral component adds up all past deviations over time and corrects for permanent steady-state errors. The differential component predicts future deviations based on the rate of change of the deviation to react quickly. These three components work together to efficiently control the system to the desired setpoint, minimize fluctuations, and ensure stable control. Figure 8 shows a block diagram of sequential and parallel PID controllers.

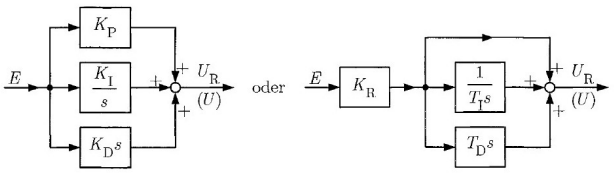

Figure 8. Blockdiagram of PID-Controllers[24]

Due to an absence of pertinent documentation and source code, it proved unfeasible to ascertain the functionality of the PID controllers of the TIAGo. The configuration of these controllers is purportedly facilitated by assigning values to P, I, and D. However, the documentation conspicuously lacked any explicit explanation concerning the correlation between these designated values and the established values K_P, T_I, and T_D, which are commonly recognized within the domain of control engineering.

4.2.3. Final implementation

Finally, the source code of the existing *arm_controller* was thoroughly researched, and a custom controller was developed based on this research. The basis for this controller is the *JointEffortController* controller type from ROS. This controller receives a target angle, calculates the deviation from the current angle as an error value, and transfers this value to the PID controller. The settings of the PID controller were adopted from PAL. It was observed that variations in these settings resulted in deterioration, manifesting as rocking or sluggish movements.

A notable distinction between the custom controller and the original *arm_controller* pertains to the transmission of angles to the PID controller. Specifically, the angles are transferred in a unfiltered and high-resolution manner. In contrast, the original *arm_controller* integrates various modules that function to check and filter values. The positions are then compared with the internal geometry model to avoid collisions, a process that also results in a reduction of resolution, as detailed in chapter 4. However, given that the controllers in this project are designed to operate within a limited, predefined range, the collision avoidance functions were deemed obsolete.

The functionality of the own controller and its integration into the *controller_manager* were successfully simulated in Gazebo. However, it was not possible to determine concrete numerical values for the current position or orientation of individual joints of the manipulator. Therefore, it was necessary to restrict the study to visually detecting movements. To this end, a publisher, *thk_accuracy*, was developed that uses x-y coordinates to move the arm a small step at a time in the direction of an axis. The visibility of these movements was then observed, i.e., whether the gripper moved appropriately, through testing with increasingly small steps, whereby movements up to a step width of 100 µm could be determined. In the digital appendix there is a video of these observations.

The attempts to implement the self-developed controller on the TIAGo were less successful, with the controller failing to produce the intended results. Even the act of loading the controller resulted in uncontrolled movements of the arm, the cause of which could not be determined. For reasons of safety, and in order to avoid any potential damage to the TIAGo, no further attempts were made with the proprietary controller on the hardware. When working with the TIAGo, the original *arm_controller* was used again despite its insufficient accuracy. The development of software for both controllers was undertaken to facilitate seamless exchange in the future; however, this endeavor necessitates meticulous research into the hardware and the interaction of disparate controllers.

4.3. Inverse Kinematic Solutions

In this project, it should be possible to specify the positioning of the manipulator using x-y coordinates. However, angle specifications are required to control the manipulator. The corresponding transformations are described below.

4.3.1. Geometric calculations

The inverse kinematics for the robot's arm was solved to accurately position the end-effector based on the identified component locations on the PCB. Figure 9 shows an overview of the different parameters that have to be transformed.

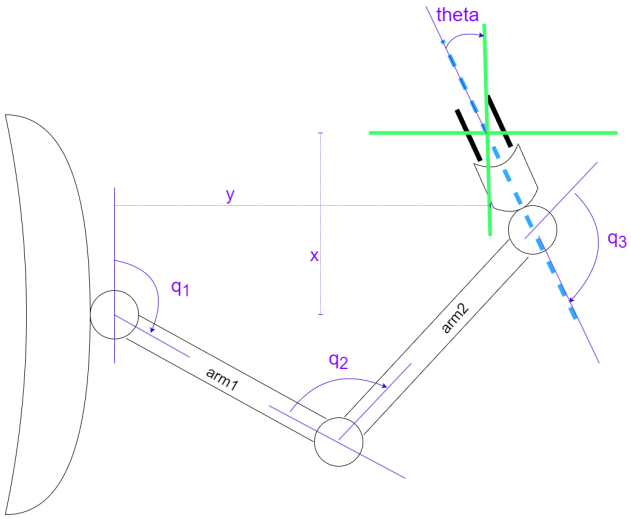

Figure 9. Simplified TIAGo arm

Calculation of joint_6 position ('wrist'):

$$x_{\text{wrist}} = x - length_{\text{arm3}} \cdot \cos(\theta) \tag{9}$$

$$y_{\text{wrist}} = y - length_{\text{arm3}} \cdot \sin(\theta) \tag{10}$$

Calculation for joint_4 angle ('ellbow'):

$$D = \frac{x_{\text{wrist}}^2 + y_{\text{wrist}}^2 - length_{\text{arm1}}^2 - length_{\text{arm2}}^2}{2 \cdot length_{\text{arm1}} \cdot length_{\text{arm2}}} \tag{11}$$

$$q_2 = \text{atan}(\frac{\sqrt{1 - D^2}}{D}) \tag{12}$$

Calculation of joint_1 angle ('shoulder'):

$$k_1 = length_{\text{arm1}} + length_{\text{arm2}} \cdot \cos(q_2) \tag{13}$$

$$k_2 = length_{\text{arm2}} \cdot \sin(q_2) \tag{14}$$

$$q_1 = \text{atan}(\frac{y_{\text{wrist}}}{x_{\text{wrist}}}) - \text{atan}(\frac{k_2}{k_1}) \tag{15}$$

Calculation of joint_6 angle (wrist):

$$q_3 = \theta - q_1 - q_2 \tag{16}$$

4.3.2. Realization of inverse kinematics

The inverse kinematic was implemented using a combined publisher/subscriber, which received information on the x-y coordinates and an alignment angle. This information is used to calculate the angles for the individual joints of the manipulator. These angles are then send to the corresponding controllers, and the movements are executed.

The development of a custom message type and its subsequent integration into ROS was imperative for the successful execution of the project.

Due to the different controllers used, two versions of this software were developed. One version, designated thk_arm_xya, employs our proprietary controller from chapter 4.2, while the other, thk_arm_xya2 utilizes the *arm_controller* developed by PAL robotics. Two complementary programs were also devised to assess and showcase the positioning and alignment capabilities, *thk_tiago* and *thk_tiago2*.

4.4. Control Algorithm Implementation and Integration

4.4.1. Implementation of Control Algorithms

Custom control algorithms were implemented to manage the robot's movements and interactions with the environment. These algorithms utilized feedback from the vision system to adjust the robot's actions in real time, ensuring precision in component placement.

To control the robot arm, a target position relative to arm joint 1 is needed, as it was described for the thk_arm_xya2 controller. Calculating this position from the image of the PCB from the head camera requires the following steps.

First the position of the PCB and its rotation relative to the camera needs to be calculated. Which is done by solving the Perspective-n-Point correspondence problem with OpenCV. This uses corresponding points on the PCB and the head camera image as input, similar to the homography calculation with the corners of the PCB contour. Solving the Perspective-n-Point correspondence problem involves calculating the extrinsic matrix shown in equation 1 which is done iteratively based on the Levenberg-Marquardt optimization [25] [26].

Second this position needs to be transformed to a position which is relative to arm joint 1. ROS includes a the package *tf* which is pre configured with all positions and of the robots joints and their current rotations and translations. Given a source frame and target frame, this tool returns the transformation between the two frames.

4.4.2. Use of the Gripper Camera

After the rough localization of the PCB and a selected component with the head camera of TIAGo, the end effector camera is used for a more precise positioning. To determine the exact position of the components center point, a cutout from the PCB reference image around this point is used. This cutout is used for template matching with the image from the end effector camera to correct the current position of the robots arm. When the template matching finds a match with a certain degree of certainty, the arm can be moved so that the component position is then in the center of the image. This process

could possibly be repeated a few times to refine the current position of the gripper. The final position can the be stored and used together with the offset of the gripper to the image center, to place the selected component on the PCB.

4.4.3. Integration with ROS

The entire control system was integrated into the ROS ecosystem, allowing for seamless communication between the vision system, kinematic solvers, and motion controllers. This integration facilitated the synchronization of sensory inputs and mechanical outputs, crucial for the accurate execution of PCB assembly tasks.

To connect the control system which was implemented as a python script with ROS instance running on TIAGo the *roslibpy* library was used. It uses the *rosbridge* package running in ROS to publish all topics in a format compatible with the *roslibpy* library. The advantage of this approach is that the control system does not have to be compiled as its own ROS Package and be deployed on TIAGo. Additionally this enables the script to be executed on a separate computer with a GPU which can significantly speed up any Image Processing and especially the machine learning based algorithms.

Both images from head camera as well as the gripper camera are received by subscribing to corresponding ROS topics with a custom callback function. But as the image processing itself is far slower than the framerate of the cameras, the callback function only stores the images in a cache. The cache can than be accessed by the image processing algorithm when it is ready to process the next frame. For the transform from the head camera frame to the frame of arm joint 1, a special topic can be requested to stream the latest transform values based on the current position of the head. The results from this topic are then stored in a cache just like the images.

If a specific component gets selected for placement, the necessary transformations are applied to the components position on the PCB. The resulting positions x and y values can then be published to the *thk_tiago_xya* topic for the *thk_arm_xya2* controller.

4.4.4. Testing and Validation

The setup underwent testing to validate the accuracy of component placement and the reliability of the robotic system. Adjustments have to be made based on the testing outcomes to optimize the system's performance.

The position relative to the camera was measured and found to have a deviation of a few centimeters, with the main error being an offset of roughly additional 4 cm in the depth. Possible causes could be the detected contour being to small or errors in the camera calibration matrix.

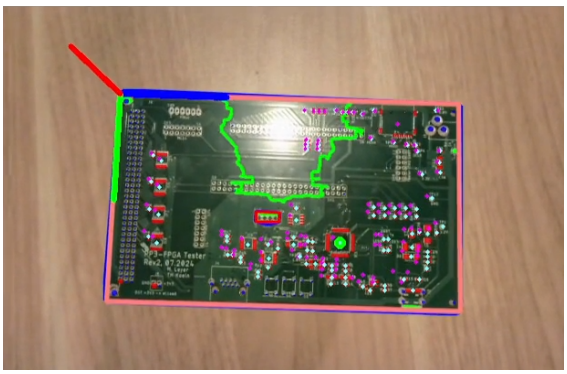

Figure 10. PCB Detection with Overlay

Figure 10 shows the Detection of the PCB outline and the projected Overlay of the PCB.

Even though the position relative to the camera is accurate to a few centimeters, the position relative to the arm, after applying the corresponding transform, is often off by more than 10 cm. The cause of this problem was not yet identified and requires further testing.

With a stable image, the template detection of the end effector camera can achieve a deviation that is smaller than the sizes of the pads on the PCB which may be accurate enough for the reflow soldering process. Because the process of aligning the arm with template matching does not rely on any detection algorithms and transformations, but directly on the matching of the images, theoretically a similar accuracy could be achieved with the robot arm. In practice the accuracy is still limited by the motor controllers and the rigidity of the arm itself.

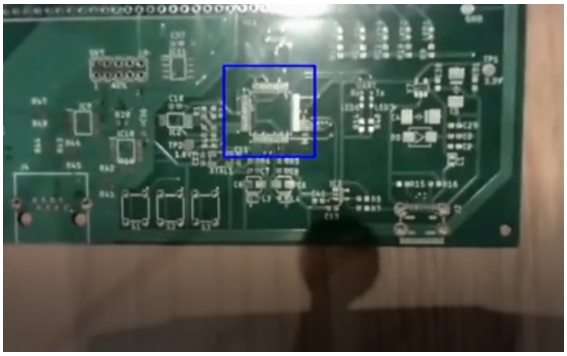

Figure 11. PCB Cutout Template Matching

An example of the template matching is displayed in Figure 11.

4.5. Conclusion

This study confirms the viability of deploying the TIAGo robot for precision tasks in electronic manufacturing. By integrating advanced vision systems, kinematic solutions, and ROS-based controls, we have demonstrated its potential in PCB assembly. While challenges remain in achieving SCARA-level accuracy, the findings highlight the adaptability of service robots in industrial applications, paving the way for further optimization and development.

5. Results

5.1. Experimental Results

The evaluation of the TIAGo robot for precision PCB assembly was conducted using a combination of visual perception, kinematic control, and robotic manipulation techniques. The primary objectives of the experiments were to assess placement accuracy, repeatability, and component detection performance.

5.1.1. Placement Accuracy

The placement accuracy of the TIAGo robot was assessed by analyzing the deviation between expected and actual component positions. Preliminary results indicated a depth offset of approximately 40 mm. This offset primarily resulted from inaccuracies in transformations between the head and gripper cameras. While template matching with the gripper camera improved alignment, precise placement remained challenging due to hardware limitations and movement inaccuracies.

5.1.2. Component Detection Performance

The component recognition capability was evaluated using feature-based detection and template matching techniques. ORB feature matching provided keypoint localization but was affected by lighting conditions and reflections. Template matching, leveraging pre-captured component images, showed promise for identifying standard PCB elements, though further improvements are needed for robustness. Machine learning-based object detection methods, such as YOLO, were not implemented due to dataset constraints and lack of validated integration within the TIAGo system.

5.1.3. Motion Control and Stability

Motion control was assessed using the existing ROS-based controller and PID control strategies. The controller allowed smooth arm movements, but software constraints due to the use of ROS, introduced limitations. The ability to consistently execute fine-grained movements below 0.01 m could just be simulated but not tested in hardware. Observations suggested that small incremental adjustments did not always produce the expected positional changes.

5.2. Discussion

The results indicate that while the TIAGo robot is capable of executing PCB assembly tasks, its precision and repeatability do not match those of industry-standard SCARA robots. Several key challenges and insights emerged from the study:

5.2.1. Vision-Based Limitations

The reliance on monocular and stereo vision for component alignment introduced significant depth perception errors. Stereo-based depth estimation exhibited high variance at close ranges, which made precise placement difficult. Due to time and resource constraints, alternative depth-sensing methods, such as structured light scanning, were not tested but remain a potential avenue for improvement.

5.2.2. Robotic Control Constraints

Despite implementing a custom ROS controller, achieving consistent and high-accuracy placement proved difficult. The robot's built-in PID controllers lacked sufficient transparency for optimization. Additionally, the build-in software framework limits accuracy in opposite to safety before programming faults.

5.2.3. Future Improvements

To enhance precision and mitigate observed errors, future work should focus on:

- Improving camera calibration and refining homography transformations.
- Exploring alternative component detection methods with validated real-world testing.
- Investigating adaptive control mechanisms for real-time error correction.
- Conducting more controlled experiments with high-precision measurement tools to quantify mechanical stability.
- Figuring out how to gain a more direct control of the TIAGos hardware.

While the TIAGo robot presents potential for flexible automation in PCB assembly, further testing and refinements are required before it can match the accuracy of industrial robotic systems.

6. Conclusion

This study explored the feasibility of using the TIAGo robot for precision tasks in PCB assembly, highlighting both its capabilities and inherent limitations. While the integration of advanced vision systems, kinematic solutions, and ROS-based control strategies enables a degree of automation, the results indicate that the TIAGo does not match the precision and efficiency of dedicated

industrial robots such as those examined in[1].

The results confirm that SCARA-based systems remain superior for high-precision PCB assembly in terms of placement accuracy, speed, and reliability. The observed depth estimation errors, motion control constraints, and software limitations of the TIAGo system reinforce this conclusion.

Future work should focus on mitigating these issues by enhancing control algorithms, adding joints on the gripper for a more precise placement, improving real-time feedback mechanisms, and leveraging machine learning for adaptive corrections. However, given the current constraints, the TIAGo remains a less optimal choice compared to specialized industrial solutions.

A. Appendix

A.1. Project Files and Documentation

The following subsections provide an overview of the project structure, including code, documentation, and additional resources. Only the functional code has been included in the appendix. For access to experimental prototype code and alternative methods, please contact the corresponding author.

A.1.1. Project Repository Structure

The main project directory contains several key folders:

- **ams_ws**: Contains workspace files.
- **Components Detection**: Includes scripts for detecting PCB components.
- **docs**: Contains project documentation.
- **Gerber_Detection**: Includes Gerber file processing scripts.
- **pcb_detection**: Scripts and methods for PCB recognition.
- **ros**: ROS-related configuration and code.
- **training_writing_detection**: Related to machine learning models for text detection.

A.1.2. Code and Additional Resources

The source code, video demonstrations, and README file are available at:

https://ilu.th-koeln.de/ilias.php?baseClass=ilExerciseHandlerGUI&ref_id=555699&cmd=showOverview
This link refers to the *AMS report and project documentation* folder of Mark Leyer as the teams final delivery.

A.1.3. Readme File

The `README.md` file in the project root provides an overview of the project, installation instructions, and usage guidelines.

A.2. Additional Figures

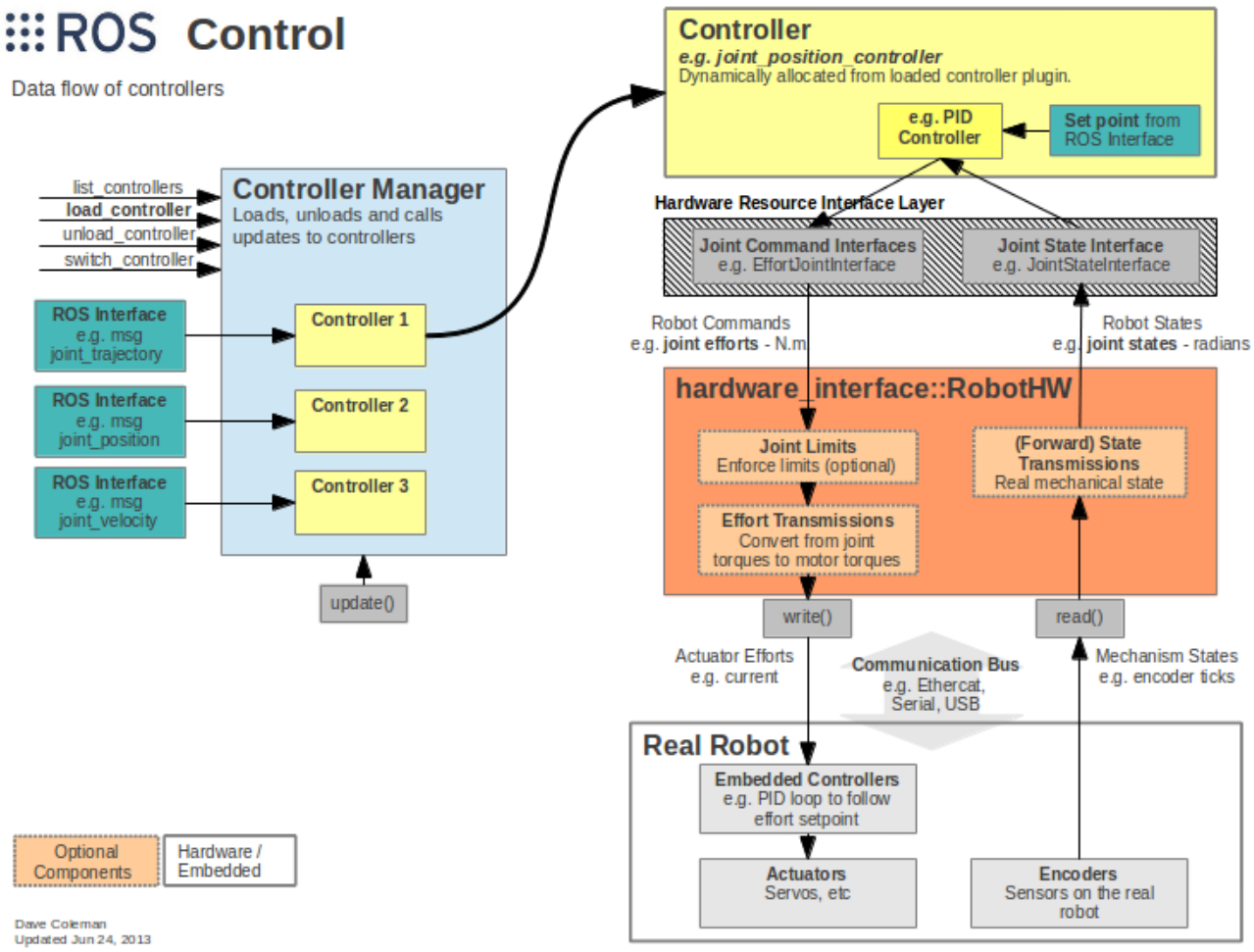

Figure 12. Overview of ROS control [27]

A.3. Project Videos

The following videos demonstrate the system in action:

- **TIAGo_initialize_arm.mp4**
  TIAGo moving arm in horizontal position.
- **TIAGo_move_arm_xya.mp4**
  TIAGo demonstrating positioning
- **TIAGo_Accuracy_100um.mp4**
  Gazebo simulation of accuracy with 100 µm step size
- **IC_Detection.mp4**
  Detection of the position of ICs and recognition of Text.
- **IC1_Position_Detection_cropped.mp4**
  Detection of pad and IC positions on a PCB.

These videos can be accessed at the provided project repository link.

List of Figures

References

[1] A. S. Nimon *et al.*, "Precision evaluation of large payload scara robot for pcb assembly," *ASME Conference Proceedings*, 2022.

[2] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 666–673, 2000.

[3] M. Strobel and T. Majetic, "Short-range depth estimation for robotic assembly tasks," *Robotics and Automation Letters*, vol. 3, no. 4, pp. 3440–3447, 2018.

[4] PAL Robotics, *TIAGo Sensor Visualization Handbook*, 2024. Accessed: 2025-02-01.

[5] J. Šmísek, M. Jancosek, and T. Pajdla, "3d with kinect: Real-time dense 3d reconstruction for rgb-d cameras," *Pattern Recognition Letters*, vol. 34, no. 7, pp. 811–820, 2013.

[6] S. Mattoccia, "Stereo vision: Algorithms and applications," *Computer Vision and Image Understanding*, vol. 162, pp. 1–13, 2017.

[7] 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.

[8] R. A. N. et al., "Dtam: Dense tracking and mapping in real-time," in *IEEE International Conference on Computer Vision (ICCV)*, pp. 2320–2327, 2011.

[9] H. Hirschmüller, "Stereo processing by semi-global matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008.

[10] R. Documentation, "Depth image topics and data streams in ros," 2022. Available: https://wiki.ros.org/depth_image_proc.

[11] F. Steinbrücker, J. Sturm, and D. Cremers, "Real-time visual odometry from dense rgb-d images," in *IEEE International Conference on Computer Vision Workshops (ICCVW)*, pp. 719–722, 2011.

[12] A. Dai, M. Nießner, and M. Zollhöfer, "Shape completion using 3d deep learning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5828–5837, 2017.

[13] J. Park and A. Khoshelham, "Depth estimation for robotic perception using sensor fusion," *Journal of Robotics and Autonomous Systems*, vol. 152, p. 104019, 2022.

[14] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[15] A. Kadyrov and M. Petrou, "Contour approximation via fourier descriptors," 2018. Accessed: February 10, 2025.

[16] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley, 2009.

[17] TheAILearner, "Perspective transformation in computer vision," 2025. Accessed: February 10, 2025.

[18] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 ed., 2003.

[19] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "Orb: An efficient alternative to sift or surf," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2564–2571, 2011.

[20] Roboflow, "Custom object detection dataset," 2025. Accessed: 2025-02-10.

[21] Ultralytics, "Yolo: Scalable object detection architecture," 2025. Accessed: 2025-02-10.

[22] JaidedAI, "Easyocr: Optical character recognition (ocr) with deep learning," 2023. Available at: https://github.com/JaidedAI/EasyOCR.

[23] PAL Robotics, *TIAGo Handbook*, 2024. https://docs.pal-robotics.com/tiago-single/handbook.html?highlight=depth#arm Accessed: 2025-02-01.

[24] H. Unbehauen, *Regelungstechnik I*. Vieweg-Teubner, 15 ed., 2008.

[25] O. S. V. Foundation, "Pose computation overview," 2025. Accessed: February 10, 2025.

[26] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[27] ROS Community, "ros_control - a set of controller interfaces for robots using ros," 2025. Accessed: February 10, 2025.