Czech Technical University in Prague
Faculty of Electrical Engineering

Technical Report

Rearranging cubes using a camera and a robotic manipulator

Robotics (B3B33ROB1)

by

Oleh Borys, Bohdan Zhytnik

Prague, Czech Republic
January 18, 2025

Contents

List of Figures

Abstract

This report presents the development and implementation of an algorithm used to calibrate and operate both a camera and a robotic manipulator to detect and rearrange cubes. The primary goals of this project were:

- To calibrate the camera and the robot effectively in order to detect cubes precisely and make the robot movements precise as well
- To generalize the solution, making it easily adaptable to different conditions of task
- To perform all operations safely and robustly, ensuring that the robot is predictable

We programmed the robot to detect ArUco markers on boards with cubes, calibrate the camera and robot, and rearrange cubes between two boards efficiently. The system accounted for spatial challenges, such as tilted boards and varying heights, by dynamically adapting the robot's approach based on the cube positions and orientations. Testing demonstrated precise detection, reliable manipulation, and safe rearrangement of cubes. These results validate the effectiveness of our algorithm.

Chapter 1

Introduction

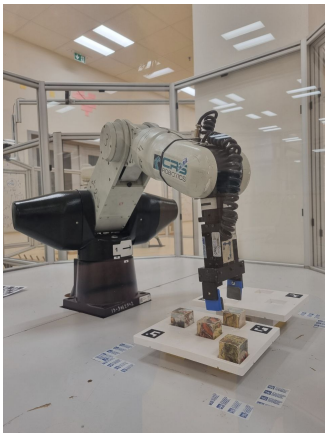
1.1 Working environment

We have a robotic manipulator **CRS A465** that was provided for us in a closed working cage with a camera inside looking down on the working space, Fig. 1.1. The robot has **6 axes** of rotation, is driven by **Mars** driving unit, and uses a Python environment for development, [1].

Also, we were provided with the following items:

1. Two boards, each of which has two Aruco Markers on opposite corners and 4 placeholders for cubes.
2. Cardboard cubes with a side approx. of 4 cm.
3. Board stands of different heights and tilts. Also, they have sponges on the bottom, making the performance saver if the robot hits the board.
4. Charuco Boards of various sizes and configurations, as well as single Aruco markers.

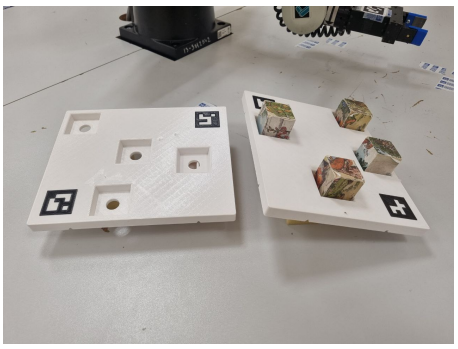
You can see our equipment and different working space configurations in Fig. 1.2.



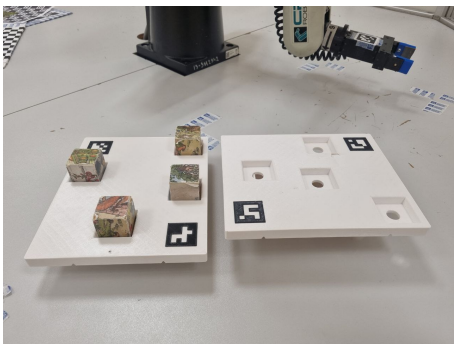
(a) The robot CRS A465

(b) The camera

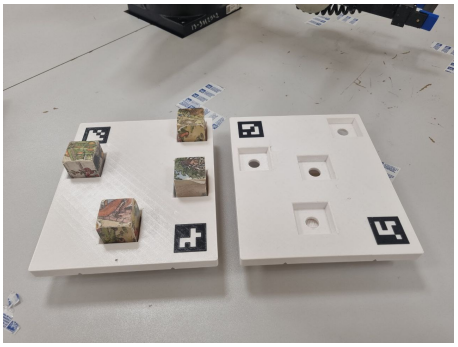
Figure 1.1: The robot and the camera in the working space



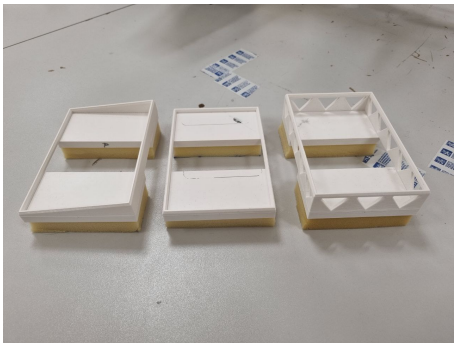
(a) Space configuration 1



(b) Space configuration 2



(c) Space configuration 3



(d) All available stands

Figure 1.2: Equipment used for the task, as well as examples of working space configurations with boards: (a) - (c); (d) - all available stands for boards

1.2 Task requirements

1. **Task Overview**
   Primarily we could divide our task into two subtypes: The **Type 1** is to take the cubes from a board that is placed parallel to the camera with a predefined height offset from the table and to place them anywhere on the table. The **Type 2** involves transferring cubes between two boards, accounting for differences in height, as well as tilts of **7** or **15 degrees**. Position of holes where cubes could be placed is predefined in **.csv** file that we get along with the board itself. The example of the board configuration is shown in Fig. 1.3. The goal in both cases is to handle the cubes safely, precisely, and efficiently.

   Here is the overview of the task of **Type 2**:

2. **Initial Setup**

   - **Board Placement**: Boards are positioned by the instructor within the robot's workspace.
   - **Calibration**: The camera and robot are calibrated to detect board positions and align movements.

3. **Cubes Detection and Pickup**

   - The robot identifies cube positions and adjusts to board tilt and height.
   - Cubes are picked up carefully, ensuring safety and precision.

4. **Transition and Placement**

   - Cubes are transferred smoothly to the target board.
   - The robot adapts to tilt and height differences for accurate placement.

5. **Completion and Validation**

   - The robot prepares to repeat points **3** and **4** until all **4 cubes** are transfered to the other board.
   - The robot returns to a safe home position.

We used the same approach for the task of **Type 1**, except for the difference that we needed to place cubes anywhere on the table and not on a target board.

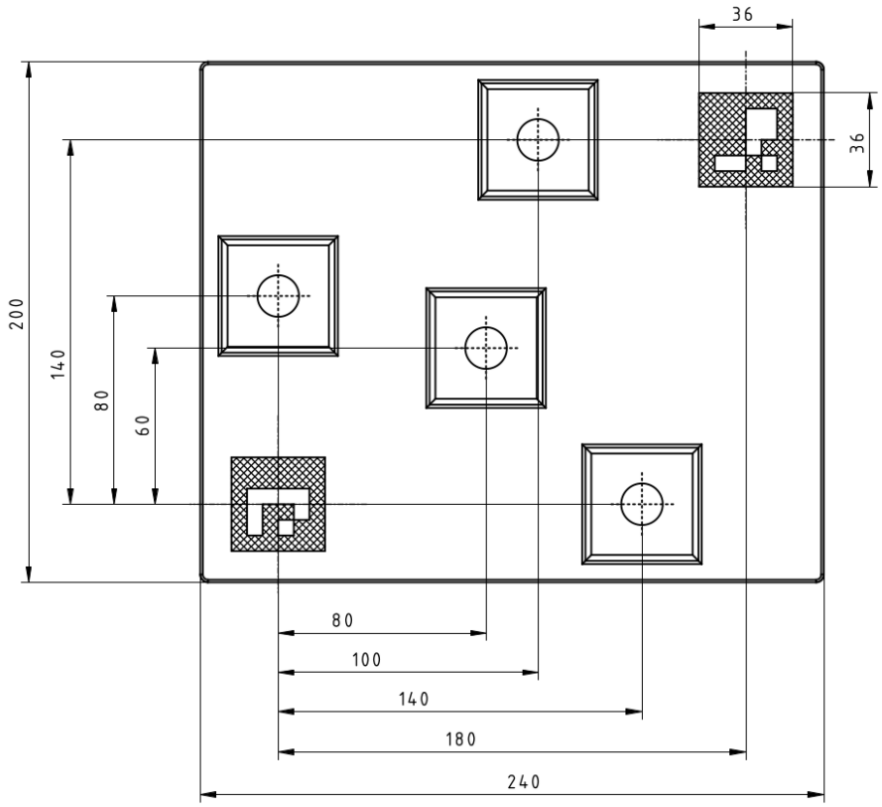


Figure 1.3: Example of board dimensions given in the board configuration .csv file

Chapter 2

Implementation

2.1 Camera Calibration using Charuco Board

2.1.1 Description of the approach

In our project, we aim to solve the problem of accurate camera calibration for detecting and estimating the position and orientation of objects in the robot's workspace. This process is critical for ensuring that the camera's intrinsic and extrinsic parameters are accurately determined.

In order to solve not only planar but space problems as well, we have chosen a method known as **3D Camera Pose Estimation with Planar Patterns**, which uses a planar board with a pattern, [2], in order to estimate 3D camera pose.

Our goal for the calibration is to find the intrinsic parameters **K**, distortion coefficients **d**, the **RMS** projection error, and the extrinsic parameters (rotation and translation vectors, **rvecs** and **tvecs**) for each image used in the calibration process. The camera matrix **K** and the camera distortion coefficients **d** mathematically represent the camera's optical properties, [3].

2.1.2 The use of Charuco board

We used a Charuco board, [4], in order to calibrate the camera, Fig. 2.1, with these specific parameters:

- Aruco marker encoding size: 5×5
- Aruco marker size of side: 22 mm
- Square size of side: 30 mm
- Width in *squares*: 7 squares
- Height in *squares*: 5 squares

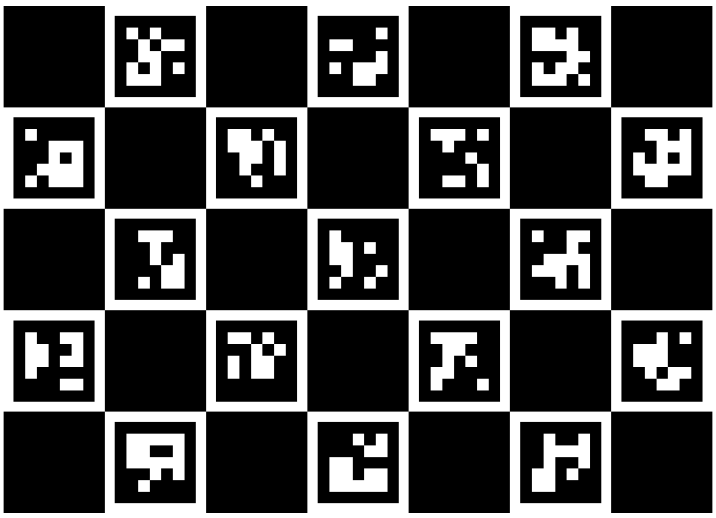


Figure 2.1: Pattern of Charuco board used for the camera calibration

We were taking photos of the Charuco board trying different poses of the board, and finally, we used **20** best photos for the calibration and came to this board pose configuration rules, that made the calibration the most accurate:

- **Consistency in Conditions**: Ensure that photos were taken on the same day, with a short time interval. Different factors such as lighting conditions in the room, camera temperature, and others can affect the camera's ability to detect objects.
- **Visibility**: Ensure that as much of the Charuco board as possible is visible in the camera frame, including a sufficient number of ArUco markers and their corresponding checkerboard corners.
- **Diverse Orientations**: Capture images of the board from a variety of angles (e.g., tilted, rotated, or partially visible) to ensure the calibration is robust to perspective distortions. But do not tilt too much to the point when Aruco markers are highly distorted, as it can increase the **RMS** reprojection error by a lot.
- **Distance**: Position the Charuco board at different distances from the camera to capture the full range of depth perspectives, from close-up to far away, within the working range of the camera. But still keep the board in focus, meaning not moving the board too close to the camera.
- **Board Coverage**: Avoid centering the board in all images. Move the board across different regions of the image (corners, edges, and center) to account for lens distortions.
- **Lighting**: Ensure proper lighting conditions. Avoid reflections or shadows that could obscure the markers or corners on the board.

2.1.3 Finding matrix K and distortion coefficients d

We used the `calibrateCameraCharuco` from `OpenCV` to estimate camera matrix **K**, distortion coefficients **d** and **RMS** reprojection error using a set of detected Charuco board corners, their corresponding IDs, and information about the physical dimensions of the board.

The resulting camera matrix **K** is presented in the Eq. 1.

$$\mathbf{K} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 4746.48725 & 0 & 1072.72375 \\ 0 & 4738.03233 & 533.49256 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

where f_x and f_y are the focal lengths in the x and y directions, γ is the skew coefficient between the axes, and (c_x, c_y) represents the principal point (optical center) of the image.

The resulting distortion coefficients $\mathbf{d}$ are presented in the Eq. 2.

$$\mathbf{d} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} = \begin{bmatrix} -0.21867 & -1.00296 & 0.00005 & 0.00311 & 8.29851 \end{bmatrix} \tag{2}$$

where k_i represents radial distortion coefficients and p_i represents tangential distortion coefficients.

The final RMS reprojection error was equal to 0.323.

2.2 Board pose and rotation estimation

Once we have the camera matrix $\mathbf{K}$ and the camera distortion coefficients $\mathbf{d}$, we can determine the position and rotation of the board with the cube holes. For this task, we solved the **Perspective-n-Points** problem, [5].

To determine the position and rotation of the board itself in the space, we detect corner positions and IDs of two Aruco markers on the board using `detectMarkers()` from `OpenCV`, then calculate the median value of their corners across **5** photos.

The reason for using the median is our observation that even if the photos were taken in a row, the values in each photo may differ slightly, which could be caused, for example, by the lighting in the room.

Subsequently, we pass corners position values from two Aruco markers together directly to the `solvePnP()` function from `OpenCV`, effectively solving the Perspective-n-Points problem of the whole board. We get the rotation vector r_{board} and the translation vector t_{board} with the origin in the center Aruco marker with the smallest ID, which matches the origin point of the coordinate system used to indicate the position of every hole on board.

Finally, we convert r_{board} to the 3×3 rotation matrix $\mathbf{R}_{camera}$ using Rodrigues' rotation formula, **??**, and calculate the transformation matrix from the hole to the camera using Eq. 3 and 4 for each hole.

$$\mathbf{h}_{camera} = \mathbf{R}_{camera} \cdot \mathbf{h}_{local} + \mathbf{t}_{camera}, \tag{3}$$

where $\mathbf{h}_{\text{camera}}$ is the position of the hole in the camera frame, $\mathbf{R}_{\text{camera}}$ is the rotation matrix from the local board frame to the camera frame, $\mathbf{h}_{\text{local}}$ is the position of the hole in the local marker frame,$\mathbf{t}_{\text{camera}}$ translation vector from the local board frame to camera frame.

$$\mathbf{T}_{CH} = \begin{bmatrix} \mathbf{R}_{camera} & \mathbf{h}_{camera} \\ \mathbf{0}^T & 1 \end{bmatrix}, \tag{4}$$

where $\mathbf{T}_{CH}$ is the homogeneous transformation matrix from the hole to the camera frame, $\mathbf{R}_{\text{camera}}$ is the rotation matrix from the local board frame to the camera frame, $\mathbf{h}_{\text{camera}}$ is the position of the hole in the camera frame.

The example of two boards with detected holes is shown in Fig. 2.2.

The $\mathbf{T}_{CH}$ matrix will be used later in order to make the robot move its gripper to the corresponding hole.

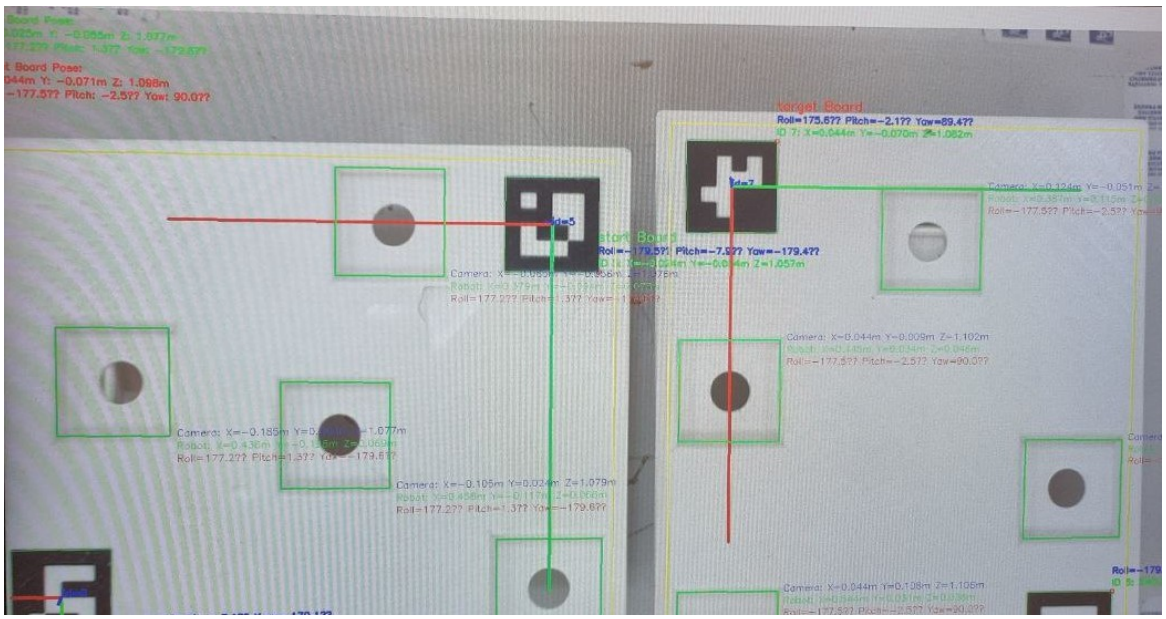


Figure 2.2: Two boards with detected Aruco markers and holes

2.3 Eye-to-Hand Calibration

2.3.1 Description of the algorithm

To obtain the transformation T_{BM}, which represents the coordinate system of the ArUco markers on the board relative to the robot base, it is first necessary to determine the transformation T_{BC}, which defines the camera's coordinate system relative to the robot base. This is achieved using the Eye-to-Hand algorithm, [6].

For this calibration, the camera must be fixed in one position within the robot's workspace. An object with an ArUco marker is attached to the robot's gripper. The object with the marker is moved by gripper to various positions such that the ArUco marker remains within the camera's field of view at each position. For each position, the transformation of the gripper relative to the robot base, T_{BG}, and the position of the ArUco marker relative to the camera, T_{CM}, must be recorded.

By providing two lists, one containing T_{BG} and the other containing T_{CM} for each gripper position in corresponding order, the transformation matrix of the ArUco marker relative to the

gripper, T_{GM}, and the transformation matrix of the camera relative to the robot base, T_{BC}, can be calculated.

It is important to note that the `OpenCV` website describes the use of the `OpenCV` library for performing Eye-in-Hand calibration. Therefore, it would be a mistake to assume that the same inputs and outputs shown on the `OpenCV` website are applicable for Eye-to-Hand calibration.

To correctly use the `calibrateRobotWorldHandEye()` function, an example from the course presentation on Robotics at CTU was followed [7].

2.3.2 Visual and mathematical representation

Below is an image visualization 2.3 of the required system consisting of the robot and the camera, along with the mathematical description of the operation of the algorithm 5.

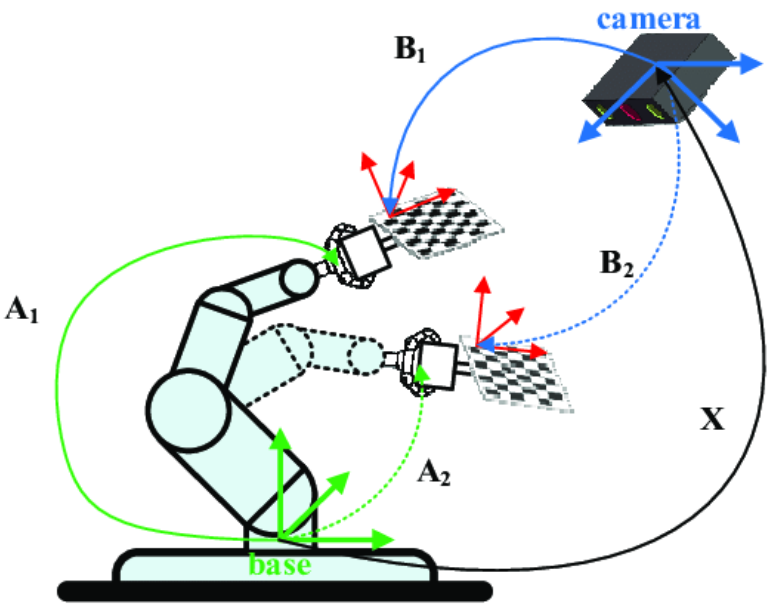


Figure 2.3: Eye to hand system example

$$\begin{aligned} T_{BC}\ T^{i}_{CM}\ T_{MG} &= T^{i}_{BG}, \\ T^{i}_{CM}\ T_{MG} &= T_{CB}\ T^{i}_{BG}, \\ T_{GM} &= T^{-1}_{MG}, \\ T_{BC} &= T^{-1}_{CB}. \end{aligned} \tag{5}$$

The system of equations is presented as follows, where T^{i}_{CM} and T^{i}_{BG} represent the transformations for the i-th position of the object with the ArUco marker. The transformations T_{GM} and T_{BC} are the outputs obtained after solving the system of equations:

To compute the required transformations, the function `calibrateRobotWorldHandEye()` from the `OpenCV` library was used. This function efficiently solves the system of equations and provides the resulting transformation matrices T_{GM} and T_{BC}.

After executing the calibration function, the following transformation matrix T_{BC}, representing the camera's coordinate system relative to the robot base, was obtained:

$$T_{BC} = \begin{bmatrix} 0.02629 & 0.99788 & -0.05940 & 0.49209 \\ 0.99942 & -0.02495 & 0.02314 & -0.01471 \\ 0.02161 & -0.05997 & -0.99796 & 1.19294 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

2.3.3 Description of selected positions

To obtain an accurate transformation matrix T_{BC}, the object with the ArUco marker must be moved to a wide range of positions relative to the camera. It is essential to include as many positions as possible with different combinations of rotations of the ArUco marker on the **x**, **y**, and **z** axes, as well as translations along these axes relative to the camera.

For the calibration process, an algorithm was used to automatically generate a list of suitable positions for the ArUco marker. Starting from an initial manually defined position 2.4, where the ArUco marker is parallel to the table, the robot rotated joints 4, 5, and 6 simultaneously to record positions with rotations of the ArUco marker around the **z**-axis. When the camera could no longer detect the ArUco marker, the robot returned to the initial calibration position, adjusted the marker's position by rotating the first joint, and updated the initial position.

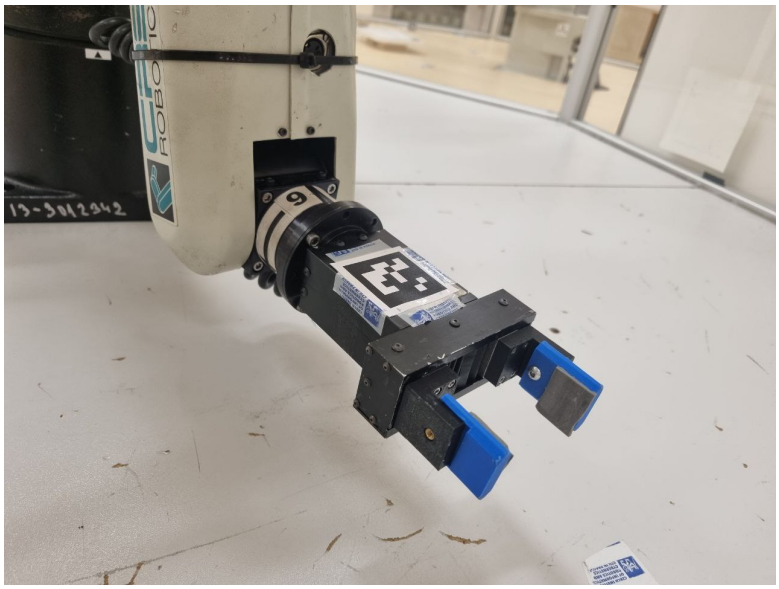


Figure 2.4: Eye to hand initial position

Additionally, after every five recorded positions, the robot slightly moved the ArUco marker forward and backward relative to the camera and applied small rotations around the **x** and **y** axes. After recording 100 positions in this manner, the robot moved 10 cm upward relative to the very first calibration position, and the cycle was repeated.

This systematic approach ensured a diverse set of marker positions, insuring the accuracy and reliability of the Eye-to-Hand calibration.

Using that algorithm a total of 270 positions were recorded.

2.4 Cube Transfer Algorithm

The robot's task has two variations. In the first, it is necessary to simply remove the cubes from the board. In the second, it is necessary to transfer the cubes from one board to another. To perform these tasks, slightly different sequences of actions are required. However, the algorithm for both tasks has a common initial part.

2.4.1 Initial Part of the Algorithms

Before running the complete algorithm, it is necessary to specify in the configuration file the type of task and the indices of the ArUco markers on the board where the cubes are initially located.

When starting the algorithm, the robot and camera are first initialized. The gripper is fully opened, and the robot moves to its initial position 2.5 (hereinafter referred to as q_{start}), where it does not interfere with the camera's operation. Then, the workspace is analyzed: the positions of the ArUco markers are determined, the positions of the cubes relative to the camera are calculated, and, in the case of a task with two boards, the positions relative to the camera where the cubes need to be transferred are calculated.

Figure 2.5: Cube Transfer Algorithm initial position

To ensure proper cube gripping, offset positions are also calculated, defined as points that are **10 cm** higher than the cube slots relative to the board with ArUco markers. In the following text, such offset positions above the cube will be referred to as "Offset."

2.4.2 Task Type 1 Execution

The task is performed in a loop repeated four times. In each iteration, the robot works with one cube. First, the position of the cube relative to the robot base is calculated. This occurs in two stages:

1. The previously obtained position of the cube relative to the camera (T_{CH}) is multiplied on the left by the transformation matrix of the camera's coordinate system relative to the robot base (T_{BC}):

$$T_{BH} = T_{BC} \cdot T_{CH} \tag{7}$$

2. Then, the rotation matrix in T_{BH} must be adjusted along the **y** and **z** axes so that the robot can correctly position the gripper for gripping the cube. For this, the alignment transformation matrix is used:

$$T_{align} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

The transformation matrix for the position of the cube relative to the robot base, which will be used for further calculations, is written as:

$$T_{BHstart} = T_{BH} \cdot T_{align} \tag{9}$$

The same two operations are performed for the Offset position corresponding to the cube, resulting in $T_{BHstartOffset}$. After this, the robot calculates the joint configurations that place the gripper at $T_{BHstartOffset}$. For this, **inverse kinematics** is used. Among all possible configurations, the one (q_{best}) with the smallest Euclidean norm difference from the current configuration is selected.

$$q_{\text{best}} = \arg\min_{q_i \in \mathcal{IK}} \|q_i - q_{\text{current}}\|_2, \tag{10}$$

where q_i is the i-th configuration from the list of configurations obtained after performing **inverse kinematics**.

The robot then applies the selected configuration.

In the same way as for $T_{BHstartOffset}$, the configuration for $T_{BHstart}$ is calculated. The robot applies the calculated configuration for $T_{BHstart}$, grips the cube, and then applies the configuration for $T_{BHstartOffset}$ again. After this, the robot rotates its first joint by 2 rad

$$q_{\text{new}} = q_{\text{current}} + \begin{bmatrix} 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \tag{11}$$

Than uses inverse kinematics to calculate the configuration for lowering the gripper with the cube by 10 cm relative to the robot base. From all the configurations, the best one is selected, and the cube is released. After this, the robot applies the q_{start} configuration.

Afterward, the positions of the cube slots relative to the camera are recalculated to ensure that the algorithm works correctly in case the robot slightly shifts the board with the cubes. The cycle is repeated the required number of times.

2.4.3 Summary of the First Task Algorithm

1. Calculate $T_{BHstart}$ and $T_{BHstartOffset}$.
2. Calculate the best robot configuration for $T_{BHstartOffset}$.
3. Apply the selected configuration.
4. Calculate the best robot configuration for $T_{BHstart}$.
5. Apply the selected configuration.
6. Grip the cube.
7. Apply the configuration for $T_{BHstartOffset}$.

8. Rotate the first joint by 2 *rad*.
9. Find the best configuration for lowering the gripper by 10 cm.
10. Release the cube.
11. Apply the q_{start} configuration.
12. Recalculate the positions of the cubes relative to the camera.

2.4.4 Task Type 2 Execution

In this task, all the same steps as in the previous task type are performed up until the point of rotating the first joint by 2 *rad*. After gripping the cube and applying the configuration for $T_{BHstartOffset}$, the robot directly applies the configuration q_{start}.

Following this, the positions of the cube slots on the second board, obtained earlier during the Initial Part of the Algorithms, are processed. For the slot, $T_{BHtarget}$ and $T_{BHtargetOffset}$ are calculated in the same manner as for the cube on the board:

$$T_{BH} = T_{BC} \cdot T_{CH} \tag{12}$$

$$T_{BHtarget} = T_{BH} \cdot T_{align} \tag{13}$$

Similarly, $T_{BHtargetOffset}$ is calculated.

After this, the cube is lowered to the target position following the same sequence of actions used for gripping the cube. Once the gripper is opened, the robot applies the configuration q_{start}.
Afterward, the positions of the cube slots relative to the camera are recalculated and the cycle is repeated.

2.4.5 Summary of the Second Task Algorithm

1. Calculate $T_{BH_{start}}$ and $T_{BH_{start_{Offset}}}$.
2. Calculate the best robot configuration for $T_{BH_{start_{Offset}}}$.
3. Apply the selected configuration.
4. Calculate the best robot configuration for $T_{BH_{start}}$.
5. Apply the selected configuration.
6. Grip the cube.
7. Apply the configuration for $T_{BH_{start_{Offset}}}$.
8. Apply the q_{start} configuration.
9. Calculate $T_{BH_{target}}$ and $T_{BH_{target_{Offset}}}$.
10. Calculate the best robot configuration for $T_{BH_{target_{Offset}}}$.
11. Apply the selected configuration.
12. Calculate the best robot configuration for $T_{BH_{target}}$.
13. Apply the selected configuration.
14. Release the cube.
15. Apply the q_{start} configuration.
16. Recalculate the positions of the cubes relative to the camera.

Chapter 3

Mistakes and Their Solutions

During the project execution, several incorrect approaches to solving specific tasks were taken.

1. One major issue was related to the recognition of ArUco markers on the boards. After developing functional code, we consistently received varying translation and rotation values for the same ArUco marker during each run of the recognition algorithm. The translation values fluctuated by approximately 10 mm, while the rotation values, represented through roll, pitch, and yaw, varied within 10° .

   The algorithm for ArUco marker recognition was rewritten multiple times until we decided to take multiple photographs and calculate the median of the translation and rotation vectors for each individual ArUco marker. Additionally, we quickly realized that calculating the median directly for rotation vectors was incorrect. Instead, the rotation vectors needed to be converted into rotation matrices, the median of the matrices computed, and then the resulting median rotation matrix converted back into a rotation vector.

2. Initially, the positions of the cube slots on the board were determined using only a single ArUco marker. This caused the robot to move the gripper to slightly offset positions when attempting to pick up a cube, often failing to grip it correctly. The solution was to use both ArUco markers on the board to determine the translation and rotation of the entire board relative to the camera.

3. Moreover, after completing the camera calibration, we initially obtained a reprojection error of around 1. As we later found out, this was a very high value. It turned out that the sheet with the Charuco board was not adequately attached to a solid surface. After securely attaching the Charuco board, the RMS reprojection error decreased to approximately to 0.323.

4. A significant mistake was made during the collection of positions for Eye-to-Hand calibration. We recorded positions using a robot gripper with an ArUco marker attached to it parallel to the gripper, Fig. 2.4. This configuration made it extremely difficult to rotate the ArUco marker along the **z**-axis relative to the robot base. Some positions were very uncomfortable for the joint limits, resulting in difficult and low-quality data collection. The marker should have been mounted perpendicular to the gripper instead of parallel, Fig. 3.1. With the marker positioned perpendicularly, it would have been possible to create more combinations of configurations by fully utilizing all 6 Degrees of Freedom (6 DoF) of the gripper.

Additionally, if a small Charuco board had been used instead of an ArUco marker for calibration, the accuracy of the calibration would likely have been improved.

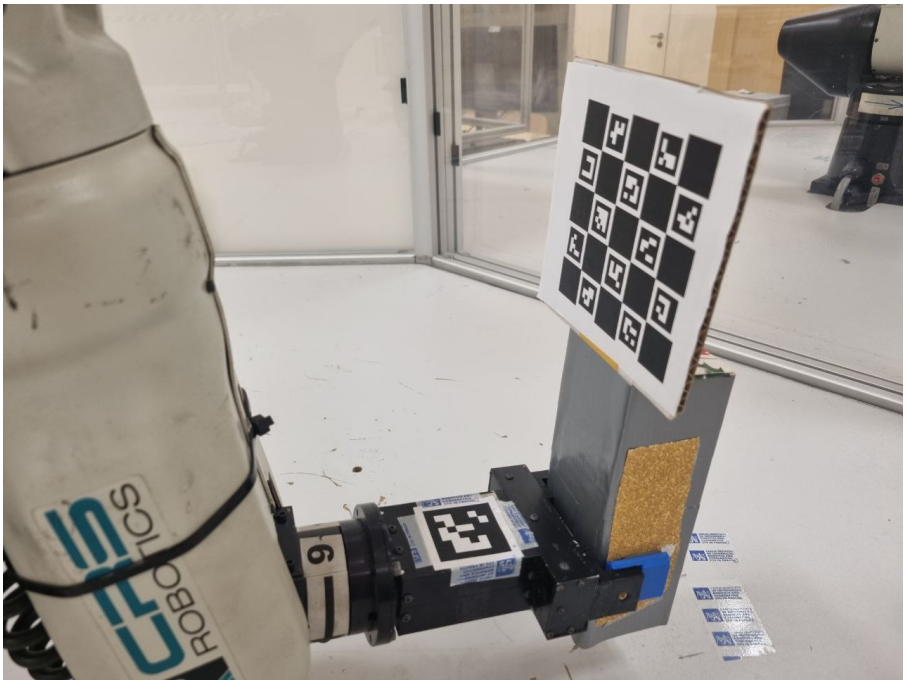


Figure 3.1: The marker that has been mounted perpendicular to the gripper instead of parallel

Chapter 4

Conclusion

The implemented project of a robotic arm for cube transfer fully meets the stated requirements. The robot is capable of removing cubes from the board and stacking them aside, as well as transfering cubes from one board to another, even when the boards are at different heights and positioned at varying angles relative to the workspace table.

We calibrated the camera using the Charuco board, finding the transformation from the marker to the camera base, and used the Eye-to-Hand algorithm to determine the camera position relative to the robot base. Subsequently, we addressed the rotation and position of the board with cube holes by solving the Perspective-n-Point problem, using the median value from 5 images to enhance robustness. Finally, we developed a cube transfer algorithm that prioritized safety, including robot speed control. For this project, `OpenCV` served as the core library.

References

[1] P.Krsek. *Robotika: Bezpečnost a práce s roboty*. 2024. URL: https://cw.fel.cvut.cz/wiki/_media/courses/b3b33rob1/krsek_rob24_robots_cs.pdf.

[2] V.Petrík. *Robotika: Základy vidění*. 2025. URL: https://data.ciirc.cvut.cz/public/projects/2024CTURoboticsData/05_perception_cz.pdf.

[3] Wikipedia. *Camera resectioning*. 2025. URL: https://en.wikipedia.org/wiki/Camera_resectioning.

[4] Wikipedia. *Chessboard detection*. 2025. URL: https://en.wikipedia.org/wiki/Chessboard_detection.

[5] Wikipedia. *Perspective-n-Point*. 2025. URL: https://en.wikipedia.org/wiki/Perspective-n-Point.

[6] Wikipedia. *Hand–eye calibration problem*. 2025. URL: https://en.wikipedia.org/wiki/Hand%E2%80%93eye_calibration_problem.

[7] V.Petrik. *Robotika: Kalibrace*. 2024. URL: https://data.ciirc.cvut.cz/public/projects/2024CTURoboticsData/05a_calibration_cz.pdf.