Czech Technical University in Prague
Faculty of Electrical Engineering

Robotics Final Project Report

Maze Navigation Using an Industrial Robot

Jakub Frgal Amir Akrami Kristýna Jurášková

16. 11. 2025

Abstract

This report presents the solution to the semester project for the course Robotics (B3B33ROB1) at the Czech Technical University in Prague. The task is to control an industrial robot equipped with a ring-shaped end-effector to navigate a maze placed on the table while respecting given constraints on time, collisions and end-effector pose. The project integrates robot kinematics, camera-based perception using an overhead Basler RGB camera, and trajectory planning. We describe the hardware setup, calibration procedures, control algorithms, experiments for different task variants (A–E), and evaluate the overall performance of the system.

Contents

1 Introduction

This report describes the solution of the semester project for the Robotics course (B3B33ROB1) focused on controlling an industrial robotic manipulator using a vision system.

1.1 Assignment Description

The assignment is to design and implement a robotic system that is able to move a contrast ring through a maze mounted on the robot's table. The maze is equipped with ArUco markers to enable robust localization in the camera image, and its dimensions are known in advance from the project repository [1].

The robot must complete different task variants, starting from Variant A and progressing to higher variants only if the previous ones are successfully solved:

- **Variant A**: Simple planar trajectory.
- **Variant B**: More complex planar trajectory.
- **Variant C**: Trajectory requiring motion outside a single plane.
- **Variant D**: Trajectory with ring orientation control.
- **Variant E**: Complex 3D trajectory with orientation control.

[2]

A trial is considered successful if the ring passes through the maze without touching its walls and finishes with a height lower than 4 cm above the base. Each attempt is limited to 2 minutes, and teams have at most 3 unsuccessful trials. Any collision of the robot with the table or with itself is disqualifying.

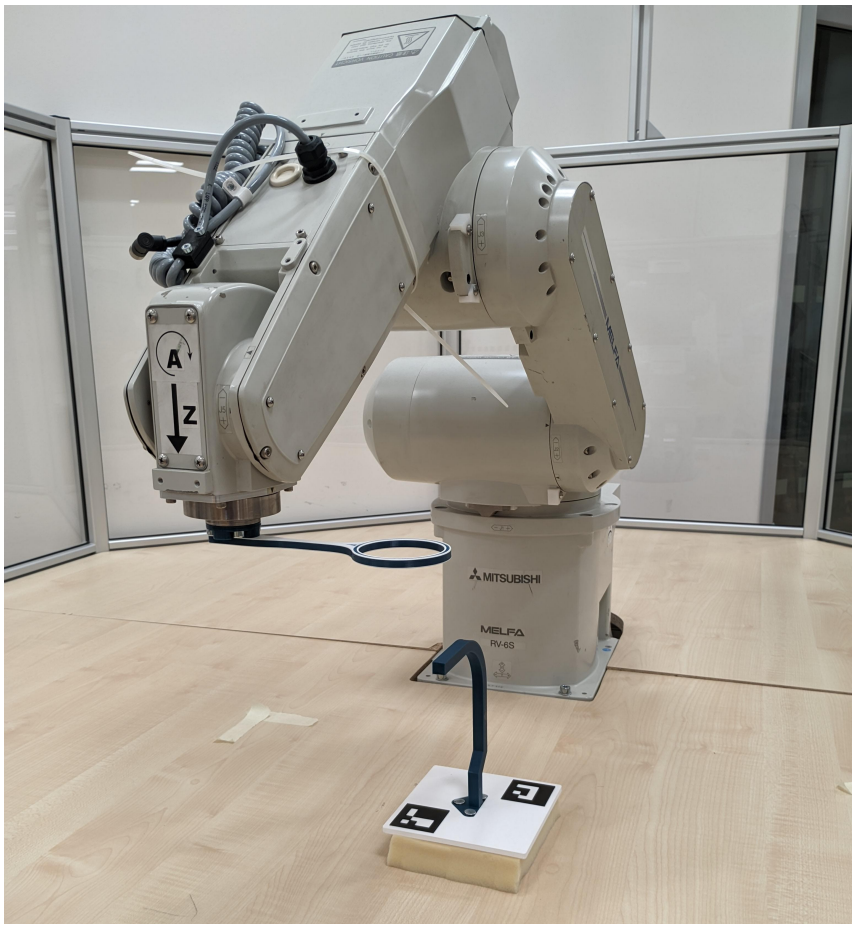


Figure 1: Robot and a maze

2 Used Equipment

In this section, we briefly describe the hardware used in the project: the robot, camera, maze, ring, and markers.

2.1 RV6S Robot

The robot used in our project is the Mitsubishi RV–6S, a six–axis industrial manipulator with a reach of 696 mm and a nominal payload of 6 kg. It is controlled via a serial connection. The robot station, including the robot controller, network infrastructure, and Python API, is part of the CTU "Robot Stations" platform used in B3B33ROB1 [3]. The official robot specification sheet is publicly available through Mitsubishi Electric [5]. The station exposes a high-level Python interface for commanding joint and Cartesian motions, which our software builds upon.

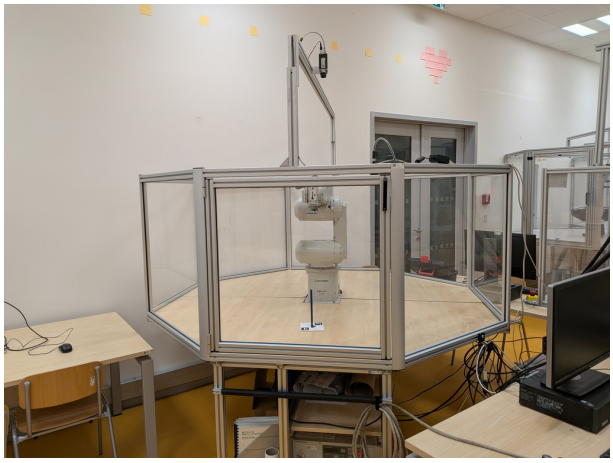


Figure 2: Mitsubishi RV-6S Robot - picture taken from [3]

2.2 Camera

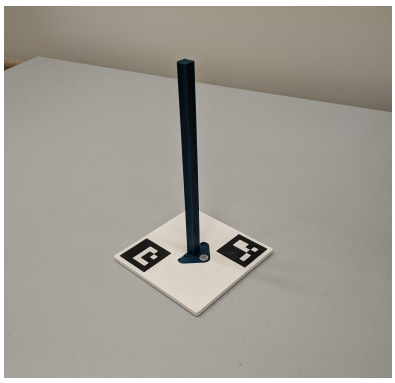
The station is equipped with a Basler acA2500-14gc camera. It is accessible over the ethernet via the name camera-rv6s. The camera was mounted above the workspace approximately perpendicular to the table plane. The camera provides images with a resolution of 2050 × 2448 pixels and is used for maze and ring detection.

Figure 3: Basler ace camera - picture from [4]

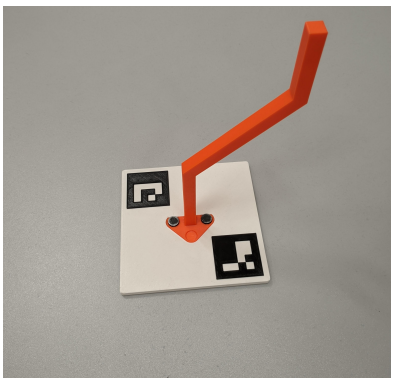
2.3 Maze

The maze is a rigid 3D structure placed arbitrarily within the robot's reachable workspace. It consists of a continuous bent rod forming a predefined spatial path that the hoop must follow. Its dimensions and geometry are provided in the official project repository [1]. The coordinate frame of the maze is defined by two ArUco markers attached to its base, which enables reliable

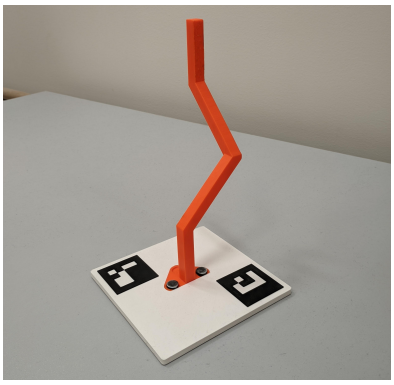
estimation of its position and orientation from the overhead camera. Five task variants (A–E) with increasing complexity are available.



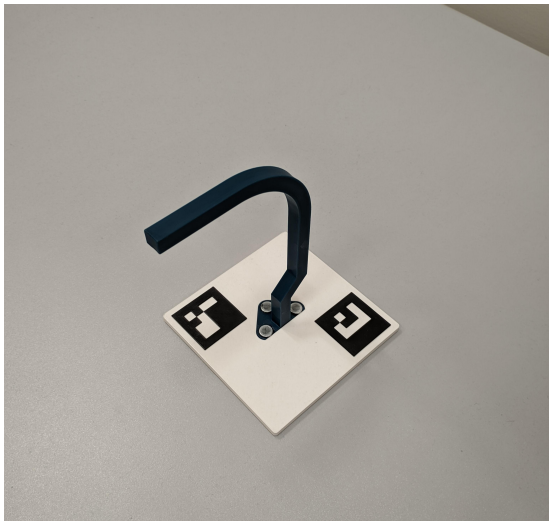
(a) Variant A



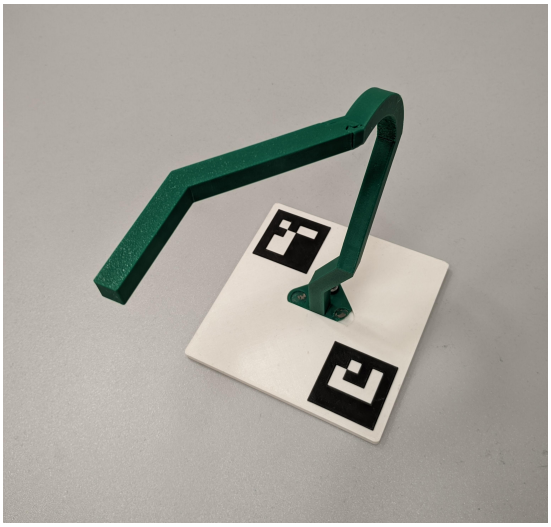
(b) Variant B



(c) Variant C



(d) Variant D



(e) Variant E

Figure 4: Comparison of Maze Variants A–E

2.4 Ring End-Effector (Hoop)

The robot end-effector carries a contrast ring (hoop) that must navigate through the maze by following the rod's geometry without making contact. The ring is rigidly attached to the robot flange, and its color and shape allow for reliable detection in the camera images. Its dimensions are documented in the project repository [1].

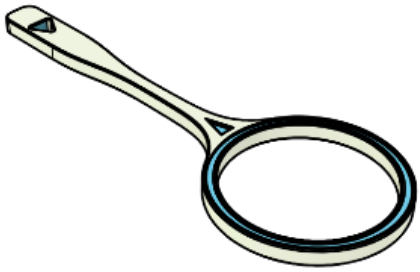


Figure 5: Hoop

2.5 ArUco Markers

ArUco markers are binary square fiducial markers designed for robust computer-vision detection and pose estimation.

For both camera calibration and maze localization we used the **ArUco DICT_4X4_250** dictionary, consisting of 4×4 bit square markers with 250 possible IDs. This dictionary was selected based on our own experimentation.

Two ArUco markers are attached to the maze base: **ID 1** and **ID 2**. When both markers are detected, their midpoint defines the maze center and their averaged orientation provides a stable estimate of the maze yaw. If only a single marker is visible, the second one is extrapolated along the known 11 cm diagonal offset between them, ensuring that the maze pose can still be reconstructed from a single camera frame.

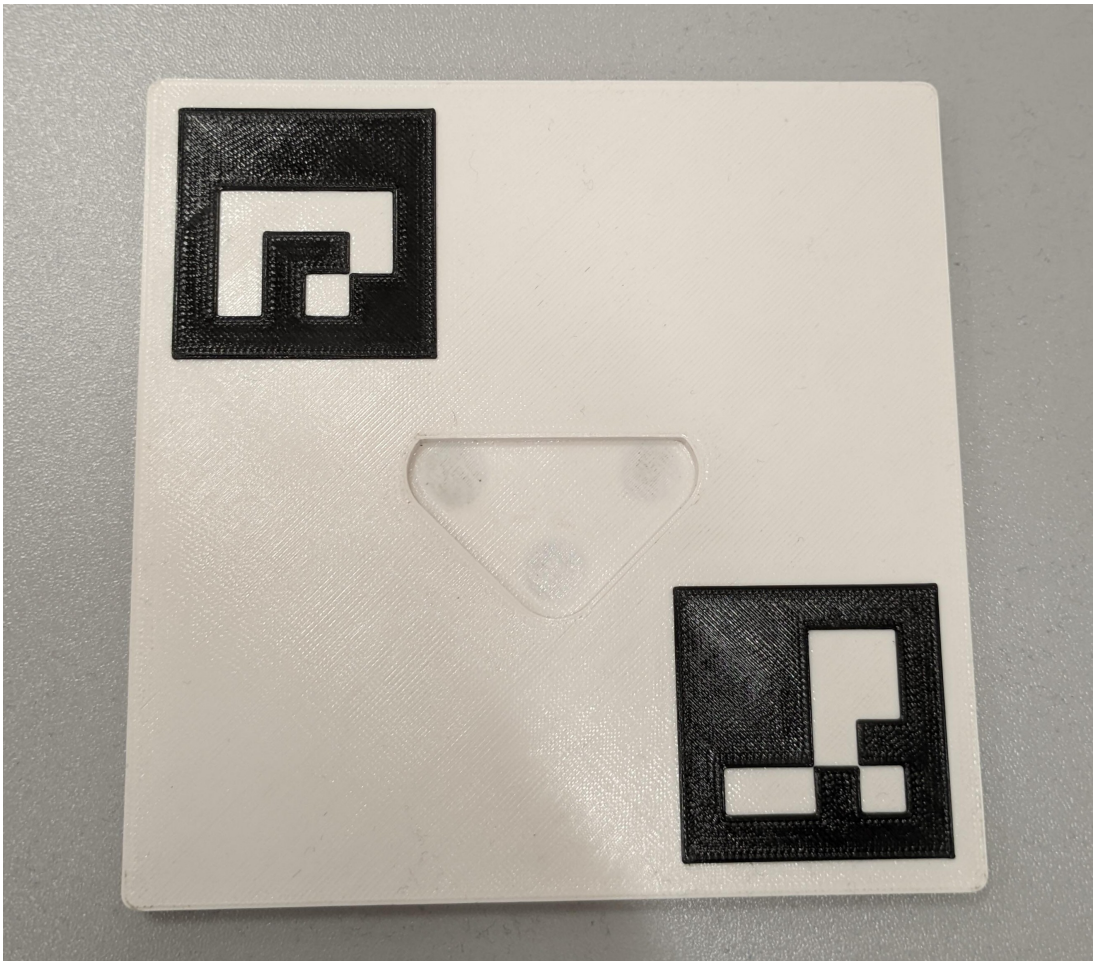


Figure 6: Maze base with ArUco markers

3 Camera Calibration

Camera calibration compensates for lens distortion and provides the intrinsic parameters necessary to map 3D points onto the image plane. We calibrated the overhead Basler acA2500-14gc camera using OpenCV and a ChArUco board.

3.1 Calibration Method

We used a ChArUco board consisting of a 5×7 grid with square side length of 35 mm and embedded ArUco markers of size 26 mm. All markers belong to the `DICT_4X4_250` dictionary, matching the dictionary used later for maze detection.

The calibration pipeline is implemented using OpenCV's `aruco` module. The dictionary, board and detector were initialized as follows:

```
dictionary = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_4X4_250)
board = cv2.aruco.CharucoBoard((5, 7), 0.035, 0.026, dictionary)
detector = cv2.aruco.ArucoDetector(dictionary)
```

Ten calibration images of the board were captured from different orientations using the Basler camera:

```
img = robot.grab_image()
```

For each image, ArUco markers were detected and the ChArUco corners interpolated:

```
marker_corners, marker_ids, _ = detector.detectMarkers(img)
_, ch_corners, ch_ids = cv2.aruco.interpolateCornersCharuco(
    marker_corners, marker_ids, img, board
)
```

Finally, the intrinsic matrix and distortion coefficients were estimated:

```
result, K, dist, rvecs, tvecs = cv2.aruco.calibrateCameraCharuco(
    all_charuco_corners,
    all_charuco_ids,
    board,
    img.shape[:2],
    None,
    None
)
```

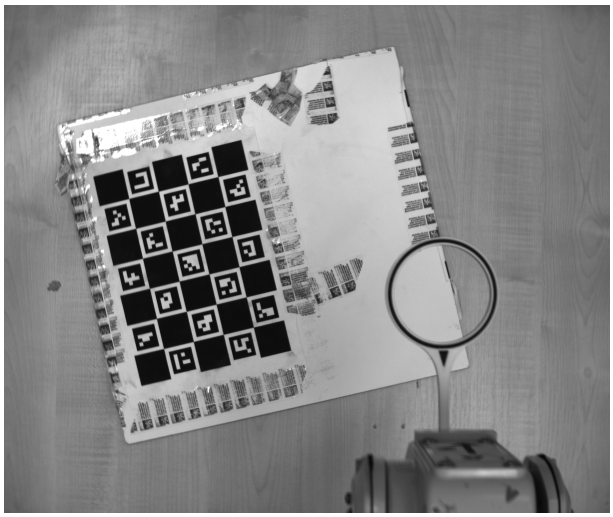


Figure 7: Example ChArUco calibration image captured by the Basler overhead camera.

3.2 Calibration Results

The resulting intrinsic matrix K of the Basler camera was:

$$K = \begin{bmatrix} 8848.10 & 0 & 1238.30 \\ 0 & 8882.88 & 1147.53 \\ 0 & 0 & 1 \end{bmatrix}. \tag{1}$$

The estimated distortion coefficients were:

$$\boldsymbol{\delta} = [-0.8325,\ -10.2931,\ -0.0158,\ 0.0022,\ 96.1910]. \tag{2}$$

These parameters are used to undistort all incoming camera images and form the basis for maze localization and hoop detection.

3.3 Hoop Detection and Planar Calibration

In addition to intrinsic calibration, we performed a separate empirical calibration to relate the pixel coordinates of the hoop to its corresponding (x, y) position in the robot base frame. This enables accurate visual alignment of the ring end-effector relative to the maze.

To obtain this mapping, the robot was moved to 20 known positions arranged on a regular 4×5 grid covering the range 0.32–0.56 m in x and -0.05–0.15 m in y. At each pose an overhead image was captured and the hoop center was detected using a Hough-circle-based method:

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (5, 5), 2)
edges = cv2.Canny(blur, 50, 150)
circles = cv2.HoughCircles(
    edges, cv2.HOUGH_GRADIENT,
    dp=1, minDist=45,
    param1=50, param2=30,
    minRadius=50, maxRadius=500
)
center_px = circles[0, 0, :2]
```

The detected pixel coordinates were paired with the corresponding robot (x, y) positions. Because the hoop lies in a plane parallel to the maze, a 2D affine model is sufficient to describe the relationship between pixel coordinates (u, v) and world coordinates (x, y). The affine transformation was estimated by least squares:

```
A, _, _, _ = np.linalg.lstsq(world_points, img_points, rcond=None)
img_est = world_points @ A      # forward mapping
```

The inverse mapping provides an estimate of the robot (x, y) position required to center the hoop at a desired pixel location (we used it to approximate image center).

Although this procedure does not estimate a full SE(3) hand–eye transformation, the empirical 2D mapping combined with the known end-effector geometry provides all information required for reliable visual alignment of the hoop relative to the maze.

We then, however, had problems with the accuracy of this automatic homography calibration. So we empiracally improved the prepcision of the camera-centering translation.

4 Robot Control and Kinematics

The robot is controlled through a high-level Python interface built around the `ctu_mitsubishi.Rv6s` API. All low-level joint commands, forward kinematics (FK), inverse kinematics (IK) and motion execution are handled by the robot controller, while the Python code implements motion primitives, pose composition, and trajectory logic.

Our contribution consists of implementing (i) custom motion primitives tailored to the ring end–effector, (ii) an improved strategy for selecting inverse kinematics solutions, and (iii) a trajectory composition module integrating calibration, perception and execution for variants A–D.

4.1 Robot Control Interface

All communication with the robot is encapsulated in the `robot.py` module. When entering the context manager, the robot is initialized, the saved calibration is loaded, and when the argument (`live=True`) the manipulator is moved to a safe "camera-friendly" pose above the workspace.

A separate script `home.py` defines a minimal homing routine returning the robot to a known configuration, while `get-current-pose.py` provides a utility for inspecting the forward kinematics pose, which we used for debugging.

4.2 Forward and Inverse Kinematics

Forward kinematics (FK) is provided by the RV–6S controller and exposed through the robotstations Python bindings. Cartesian commands are specified as poses in SE(3) and internally transformed into joint angles by the controller's built-in IK solver.

In our project, multiple IK solutions may exist for a given Cartesian pose. Directly accepting the first valid configuration may cause unnecessary wrist flips or unstable elbow postures. To mitigate this, `robot.py` evaluates all feasible IK branches and selects the one minimizing a custom "hoop path length" metric:

- the hoop center is propagated along each candidate joint configuration,
- the geometric path deviation from the current hoop position is computed,
- the solution with minimum displacement is chosen.

This ensures smooth motions across variants and avoids random jumps between kinematic branches.

4.3 Motion Primitives

Our controller firstly supports translation (`translate`) that takes in vector by which the robot relatively moves in respect to its current position.

Fixed–frame rotations. The primitives `rotate_x_fixed` and `rotate_z_fixed` implement rotations around axes attached to the world or maze frame. To keep the hoop center stationary during a rotation, a tool offset $L = 145\,\text{mm}$ (distance from flange to hoop center) is applied before and after the rotation:

```
T = current_pose
T = T @ translate(-L, 0, 0)          # shift toward hoop center
T = T @ rotate(angle, axis="x")      # apply rotation
T = T @ translate( L, 0, 0)          # shift back to flange
robot.move(T)
```

4.4 Pose Composition (Body vs. World Frames)

Higher-level motions are constructed using the `compose_movement()` routine, which accumulates elementary translations or rotations into a final SE(3) transformation. Each step is parameterized in either the **body frame** (attached to the current tool orientation) or the **world frame** (aligned with the maze and robot base).

Let the end-effector pose at step k be

$$T_k = \begin{bmatrix} R_k & p_k \\ 0 & 1 \end{bmatrix}. \tag{3}$$

A step is given by a pair (R_s, p_s, frame). Pose composition is then

$$T_{k+1} = \begin{cases} T_k \begin{bmatrix} R_s & p_s \\ 0 & 1 \end{bmatrix}, & \text{frame} = \text{body}, \\ \begin{bmatrix} R_s & p_s \\ 0 & 1 \end{bmatrix} T_k, & \text{frame} = \text{world}. \end{cases} \tag{4}$$

Thus:

- **Body-frame steps** follow the tool's orientation,
- **World-frame steps** preserve alignment with maze axes.

This dual-frame interface enables intuitive movements such as:

- translating along maze axes regardless of robot yaw,
- applying yaw corrections after vision alignment,
- rotating the hoop around its center using the offset geometry.

Listing 1: Example of multi-step composed movement

```
steps = [
    {"type": "trans", "frame": "body",  "p": [0.2, -0.3, 0.0]},
    {"type": "rot",   "frame": "world", "R": self.build_rotation("y", np
        .pi)},
]
T = self.compose_movement(steps)
self.move(T)
```

4.5 Inverse Kinematics and Motion Execution

The `move()` method transforms geometric instructions into stable joint-space motions. The pipeline is:

1. Read current joint configuration q_0.
2. Compute the desired target pose
$$T_{\text{goal}} = T_{\text{current}}\, T_{\text{offset}}. \tag{5}$$
3. Compute all feasible IK solutions
$$q_1^{(i)} = IK(T_{\text{goal}}). \tag{6}$$
4. Select the solution minimizing predicted hoop motion.

Hoop-Centered IK Selection

To ensure smooth and stable motion of the hoop during Cartesian movements, all feasible inverse kinematics (IK) solutions are evaluated using a predictive metric based on the expected 3D displacement of the hoop center.

Let

- $q_0 \in \mathbb{R}^6$ be the current joint configuration,
- $q_1^{(i)} \in \mathbb{R}^6$ be the i-th IK solution,
- $h(q) \in \mathbb{R}^3$ be the hoop center position obtained via forward kinematics combined with the known flange→hoop offset,
- n be the number of interpolation steps used to predict motion smoothness.

We approximate the joint trajectory between q_0 and a candidate solution $q_1^{(i)}$ by linear interpolation in joint space.

$$q(t_k) = (1 - t_k)\, q_0 + t_k\, q_1^{(i)}, \qquad t_k = \frac{k}{n}, \quad k = 0, \ldots, n. \tag{7}$$

Using this discretization, we compute the predicted geometric path length of the hoop center:

$$C_i = \sum_{k=1}^{n} \left\| h(q(t_k)) - h(q(t_{k-1})) \right\|. \tag{8}$$

The cost C_i measures how much the hoop would move in space if the robot transitions from q_0 to $q_1^{(i)}$. The IK branch producing the smallest displacement is selected:

$$q_{\text{best}} = \arg\min_i C_i. \tag{9}$$

This criterion strongly penalizes wrist flips and discontinuities between kinematic branches, resulting in smoother execution and more consistent behavior during all maze variants.

Listing 2: Selecting IK solution that minimizes hoop travel

```
q_candidates = self.robot.ik(new_pose)

def cost(q_target):
    return self.hoop_path_length(q0, q_target, n_steps=30)

q_best = min(q_candidates, key=cost)
self.robot.move_to_q(q_best)
```

This strategy prevents wrist flips, avoids sudden joint jumps, and keeps the hoop stable relative to the maze—especially important for Variant D, where orientation constraints are strong.

4.6 Tool Offset and End–Effector Geometry

The ring end–effector is mounted at a fixed pose relative to the robot flange. The vector between the flange coordinate system and the hoop center is known from real-world measurements and encoded in `robot.py` as a fixed offset.

4.7 Yaw Bias and MAGIC_OFFSET

During testing, we observed a consistent systematic translation misalignment between the maze coordinate frame and the robot's perceived coordinate frame obtained from ArUco detection when starting yaw_bias has been applied. The discrepancy was small but stable. To compensate, the software applies a bias correction:

$$\text{MAGIC\_OFFSET} = \frac{0.02}{\pi} \text{ cm/rad.} \tag{10}$$

This correction is added after moving above the maze start. Although empirical, this constant substantially improves repeatability and reduced drift in all variants A–D.

5 Main Program and Execution Pipeline

The main program integrates calibration, perception, planning and robot control into a unified execution pipeline. The top-level script `main.py` selects a maze variant (A–D), configures optional yaw offset/bias, initializes the robot, acquires the maze pose from the camera and executes the corresponding trajectory. All low-level motion and pose composition logic is implemented in `robot.py`.

5.1 Program Structure

When executed, `main.py` performs the following sequence:

1. Load camera calibration and initialize the RV-6S robot.
2. Move to a predefined safe observation pose.
3. Capture an image and estimate the maze pose from ArUco markers.
4. Move above the maze start in respect to selected variant with optional user bias.
5. Execute the trajectory of the selected variant.
6. Moves to home.

A simplified excerpt from `main.py` illustrates this flow:

Listing 3: Main program entry point (robot.py)

```
with Robot(live=True) as r:

    yaw_bias = bias * np.pi  # optional user bias
    variant  = variant       # "A", "B", "C", or "D"

    r.run(variant, yaw_bias)
```

5.2 Maze Pose Estimation and Pre-Alignment

Before executing any variant, the robot must compute the transformation from the robot base to the maze coordinate frame. This is done by:

1. Detecting ArUco markers 1 and 2.
2. Averaging their orientation (if both visible).
3. Computing the maze center and yaw.

Finally, the robot calls `move_to_maze_start()`, which positions the hoop at the canonical maze entry configuration. This pose is computed entirely from calibrated transforms:

$$T_{\text{start}} = T_{back}\, T_{center}\, T_{start}\, T_{magic}\, T_{rotation} \tag{11}$$

where T_{back} is the canonical entry configuration→camera-center transform, T_{center} is the camera-center→maze-center, T_{start} is the known maze entry pose, and T_{magic} with combination of $T_{rotation}$ fixed the world hoop yaw to yaw_bias specified by the user and corrects the translation error caused by yaw offseting. All motions that follow assume that the hoop starts exactly at this canonical entry configuration.

6 Variant Execution (A–D)

Each variant is implemented in `robot.py` as a dedicated method:

- `variant_A()` — straight vertical trajectory,
- `variant_B()` — single-bend planar trajectory,
- `variant_C()` — multi-bend planar trajectory,
- `variant_D()` — spatial trajectory requiring a $\pi/2$ rotation of the hoop.

Variants A–C share a common execution principle: *their geometry can be followed using pure translations while maintaining a fixed hoop orientation.* Because of that only translation is used in the solving process. Except for yaw_bias, no rotational motion of the hoop is required.

Variant D introduces a fundamentally different configuration. The overhanging portion of the maze cannot be traversed without rotating the hoop about its own axis. Therefore, Variant D combines a translational path with an orientation change of $\frac{\pi}{2}$ around the hoop center, implemented using the rotation primitives described in Section 4. This makes Variant D more sensitive to calibration accuracy and inverse-kinematics branch selection.

6.0.1 Variant A

Variant A consists of a single straight segment pointing upward from the base plate. The hoop must descend onto the start point and slide along a vertical line.

Motion description.

- Lower vertically to preset z axis position.
- Move back up.

6.0.2 Variant B

The Variant B model includes a diagonal shaped bend. This requires the hoop to translate along the two axes at once.

Motion description.

- Approach and enter the first straight segment.
- Translate along the diagonal section.
- Vertically lower to the preset position of the z axis.
- Move back up.

6.0.3 Variant C

Variant C expands on Variant B but adds one more bend. The motion is very similar to B in nature.

Motion sequence.

- Approach and enter the first straight segment.
- Translate along the first diagonal section.
- Translate along the second diagonal section.
- Vertically lower to the preset position of the z axis.
- Move back up.

6.0.4 Variant D

The variation D contains a pronounced 3D overhang that requires the hoop to rotate around its own axis. This is the only variant where orientation is essential: without rotation, the hoop cannot clear the overhanging segment.

Motion description.

- Approach the overhang.
- Perform a hoop-centered rotation of $\pi/2$ in multiple rotations and translations.
- Follow the downward and forward continuation of the rod.

7 Experiments

All experiments were carried out on 14 November 2025 at the RV-6S workstation in the CIIRC laboratory. For each trial, the maze was placed arbitrarily on the table, and its pose was estimated using ArUco markers before executing the selected variant. The same control and calibration pipeline was used for all variants (A–D).

We evaluated success rate, repeatability, and robustness against small maze pose variations. A trial was considered successful if the hoop followed the entire rod without touching it and ended below the required 4 cm height threshold.

7.1 Variants A–C

Variants A–C rely exclusively on translational motion and therefore do not require orientation control of the hoop. The trajectories consist of piecewise-linear segments aligned with the rod geometry.

Results. All three variants achieved a **100% success rate** across all executed trials. The hoop-centered IK selection completely eliminated branch-switching issues, and no collision or stability problems were observed. Even when the maze was rotated or shifted by hand between runs, the calibrated pipeline reliably adapted to the new pose.

7.2 Variant D

Variant D introduces the first orientation–dependent section of the maze. The hoop must rotate around its axis while simultaneously following a curved portion of the rod. This motion pushes the manipulator close to several of its joint limits and exposes the sensitivity of the RV-6S to certain wrist–elbow configurations.

Observed behavior. In most trials, the planned flip maneuver executed correctly. However, in approximately 10% of the runs, the IK solver selected a "bulky" posture in which the wrist and elbow extended outward more than expected. Although the pose was kinematically valid, it increased the sweep radius of the hoop, causing a slight touch on the rod during the mid-curve rotation.

These failures were not due to perception errors or calibration drift. They were entirely caused by occasional selection of less favorable IK branches during the rotation, which is a hardware-induced limitation of the RV-6S kinematic structure.

Results. The measured success rate for Variant D was therefore **approximately 90%**. All failed runs resulted from minor hoop–rod contact caused by suboptimal joint configurations during the flip, not from trajectory planning or maze localization errors.

7.3 IK Feasibility Across Variants

Although the RV-6S generally provides stable IK solutions for the maze geometry, occasional infeasible poses were encountered in all variants. These events occurred when:

- the robot was positioned near the boundary of its reachable workspace,
- the planned movement temporarily approached a wrist singularity,
- the trajectory requested a momentary posture outside the joint limits.

Such cases were rare in Variants A–C. In Variant D, however, the combined translation–rotation requirement increased the likelihood that the robot passed briefly through postures where the available workspace was narrow.

Importantly, these IK feasibility issues were not counted as "unsuccessful attempts," because they reflect fundamental hardware constraints rather than execution failures caused by the control strategy.

7.4 Decision Regarding Variant E

Variant E extends Variant D with an even more demanding spatial trajectory including multiple orientation-dependent segments. Based on the results of Variant D, where occasional bulky IK postures caused the hoop to lightly touch the rod, we concluded that Variant E would likely exhibit similar—or more severe—issues.

Since Variant E represents only a small fraction of the final grade, and given that its successful execution would require substantial additional tuning of IK behavior and trajectory reshaping, the expected return on investment was low. For this reason, and due to the intrinsic hardware limitations of the RV-6S, we decided not to attempt Variant E.

7.5 Execution Guarantees

Throughout all experiments, the execution loop enforced a consistent set of safety and stability constraints:

- **Collision avoidance:** Trajectories preserved clearance from the table and the robot body at all times.
- **IK stability:** The hoop-centered IK selection minimized unnecessary joint changes and favored stable postures.
- **IK feasibility checks:** When no valid IK solution existed for a given intermediate pose, the controller halted the motion to avoid undefined behavior.
- **Reversibility:** All experiments could be safely restarted by invoking the `home.py` script, which places the robot into a well-defined, collision-free state.

These measures ensured repeatable and reliable experimental behavior across different maze placements and multiple trial repetitions.

8 Conclusion

In this project, we developed a complete robotic system capable of navigating a 3D rod-shaped maze using a Mitsubishi RV-6S manipulator and an overhead Basler camera. The system integrates Charuco-based intrinsic calibration, ArUco-based maze pose estimation, empirical hoop-plane calibration, trajectory composition, and a custom IK-selection strategy designed to minimize predicted hoop displacement.

The system achieved **100% success** on Variants A–C, which require only translational motion. For Variant D, which introduces orientation-dependent motion, we achieved **approximately 90%** success. All failures were due to occasional suboptimal IK postures that caused minor hoop–rod contact during the flip maneuver. These effects reflect inherent limitations of the RV-6S kinematic structure rather than deficiencies in our control algorithm or calibration pipeline.

Given these hardware-induced constraints and the limited point value of Variant E, we deliberately chose not to implement the final variant. Nevertheless, the overall system performed robustly and demonstrated fast, repeatable execution across multiple randomized maze placements.

Future improvements could include optimized trajectory generation and planning to smartly search for all feasible variants of movement beforehand.

References

[1] CTU in Prague, "B3B33ROB1 Maze Project Repository," Available online: https://github.com/pilcsimo/ROB1-2025-hw.

[2] CTU in Prague, "B3B33ROB1 Final Project," Available online: https://cw.fel.cvut.cz/wiki/courses/b3b33rob1/final_project.

[3] CTU in Prague, "Robotics Robot Stations Documentation," https://robotics-robot-stations.readthedocs.io/en/latest/. Accessed: January 2026.

[4] Basler acA2500-14gc camera https://www.baslerweb.com/en/shop/aca2500-14gc/

[5] Mitsubishi Electric Corporation, "MELFA RV-6S Industrial Robot – Product Specification," Available online: https://robotics-robot-stations.readthedocs.io/en/latest/stations/rv6s.html. Accessed: January 2026.