Flexible Computer Vision based Sample Switching System using a Robotic Arm

Anthon Andersson

Anton Håkansson

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6247
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2024

Abstract

This thesis presents the development of a computer vision based robotic system for flexible sample switching at the European Spallation Source (ESS). At ESS, samples will be placed in the neutron beam by the instruments. These environments will be exposed to radiation, making direct access to the samples difficult for personnel. To maximize scientific output, it is also desirable to minimize the downtime of the neutron beam. These factors make an automated solution for sample switching highly desirable. Traditional robotic systems often lack the adaptability needed to perform in settings that are subject to change. Consequently, it is hard to develop robotic systems that are general enough to be independent of a specific robotic arm and the environment, using the traditional approach. This project addresses these limitations by creating a robotic system capable of operating autonomously in a dynamic environment.

The system integrates several components: a Stäubli TX60 industrial robot, an Intel RealSense D435if depth camera, and custom 3D-printed sample handles with ArUco markers for identification and pose estimation. ROS is used for control and perception, and EPICS drivers is developed for integration with ESS's control system.

The results demonstrate the feasibility of integrating autonomous robotic systems into complex research environments, showing promise in improving operational functionality in sample handling at ESS. Future work will focus on enhancing system robustness, upgrading hardware, expanding functionality, and further integration with ESS's control systems.

Overall, this thesis contributes to the greater field of robotics by providing a case study of a versatile and adaptable robotic system, highlighting challenges and solutions in developing autonomous systems for scientific research.

Acknowledgments

We would like to express our gratitude to the individuals who supported us throughout this project. Our academic supervisor, Björn Olofsson, provided guidance and insightful feedback. We also extend our thanks to Tomasz Bryś and Karin Rathsman, our ESS supervisors, for their continued support and assistance. Additionally, we are thankful for Jan Hrivnak's assistance with the 3D prints of the sample handles. Lastly, we thank Anders Pettersson for his help to move and get the robot running.

The thesis has been authored following funding from the European Spallation Source ERIC (ESS).

Contents

1

Introduction

Traditionally, robotic automation has been characterized by clearly defined operations within controlled and structured environments. These systems are typically engineered to execute repetitive tasks with high precision but often lack the flexibility required to adapt to dynamic or unpredictable conditions [Heyer, 2010]. This rigidity makes traditional robotic automation less suitable when developing general solutions for dynamic environments.

At the European Spallation Source (ESS), there is a need for sample switching systems to move samples into and within the neutron beam of the instruments. Currently, various specialized systems are suggested, such as the one by [Schmid et al., 2021]. These systems are designed to meet the specific requirements of different sample types and instruments. However, these solutions might prove difficult to generalize and use across different experimental setups. This project aims to overcome those constraints by developing an autonomous robotic system for a robotic arm that is both flexible and versatile, capable of operating on different samples without being limited by constraints such as a specific robot model or a specific environmental setting. The goal is to design the autonomous system in such a way that functionality can be easily added, resulting in a greater degree of versatility and adaptability.

At ESS, there is a target for 95% facility reliability, a high-level goal of the ESS facility. This goal places significant demands on the processes at ESS [Peggs, S. et al., 2012]. The robotic system under development in this thesis aims to address limitations to a specialized system by creating a solution that is flexible and general enough to be integrated into any of the fifteen instruments at ESS.

The methodology chosen for this thesis includes a literature review of the current state of laboratory automation and robotics, followed by practical experimentation to design and develop a system that can make this process more dynamic and flexible.

1.1 Background

European Spallation Source ERIC

European Spallation Source ERIC is a research facility currently being built in Lund, Sweden [*About ESS* n.d.]. It is Europe's next flagship big science project and is funded by 13 European member countries. When finished, it is set to become the most powerful neutron source in the world and represents a pinnacle to the scientific advancement of materials research. By accelerating protons to 96% the speed of light and targeting them at a rotating tungsten wheel, neutrons will be generated through spallation. These will be guided to neutron scattering instrument, enabling scientist to use the neutrons for research applications [*How the Accelerator works* n.d.]

Previous Work

In this section, several key studies that have influenced the development of this thesis are reviewed. Foundational insights into the use of robotic systems for various applications, ranging from vision-based control to sample manipulation in scientific settings, are provided by these works.

Vision-Based Autonomous Robot Control for Pick and Place This article discusses vision-based pick and place, which is highly relevant to this thesis. The authors also go in-depth into the calibration of a camera fixed in the workplace, a design choice that is similar to the approach taken in this thesis [Kotthauser and Mauer, 2009].

Intelligent Automation of Dental Material Analysis Using Robotic Arm with Jerk Optimized Trajectory Focusing on the optimization of robot trajectories to minimize jerking motions, this article addresses a critical aspect of robotic automation: the need to handle sensitive samples without causing damage. This concern is particularly relevant to the thesis, given that the integrity of samples at ESS must be preserved to ensure the validity of scientific results [Damaševičius et al., 2020].

Improved sample manipulation at the STRESS-SPEC neutron diffractometer using an industrial 6-axis robot for texture and strain analyses The study ("Improved sample manipulation at the STRESS-SPEC neutron diffractometer using an industrial 6-axis robot for texture and strain analyses") by Randau et al. presents an examination of the utility and adaptability of industrial 6-axis robots in scientific laboratories. It highlights the robot's versatility in handling diverse sample sizes and weights, offering manipulation and automated sample changing capabilities, which is important for complex experimental arrangements [Randau et al., 2015].

1.2 Questions and Goals

As can be seen in [Randau et al., 2015], robotic automation is used as sample manipulators in complex research environments. This solution is implemented in the more traditional way, using the robot language specific to the manufacturer. Similarly, they have written scripts that translate the manufacturer-specific language into the language of the control system. However, since the control system only accepts singular commands, they have had to simplify the robot's movements significantly. This means that if they were to add more functionality or use the system on another robot, they would have to invest significant time in developing that capability.

This thesis will investigate whether there is a way around these limitations by developing a system that is flexible in both its operation and its compatibility with different robots and environments. The goal is to create a solution that enhances functionality and adaptability without requiring extensive reprogramming or system modifications.

- **Research questions**
  - **Robotic Automation and System Integration in Research Applications** - How can robotic automation, specifically through the use of industrial robots, be effectively integrated into research applications for manipulating samples?
  - **Precision and Efficiency in Robotic Operations** - What levels of precision and efficiency can be achieved in robotic operations related to the placement and movement of samples within scientific settings?
  - **Recognition and Positioning Techniques for Robotic Systems** - How can recognition and positioning techniques be applied within robotic systems to enhance the identification and manipulation of samples in research and industrial environments?
- **Goals for the European Spallation Source**
  - **Introduction of Robotics Knowledge to ICS** - To facilitate the transfer and implementation of robotics knowledge at the Integrated Control System (ICS) domain at ESS.
  - **Feasibility for EPICS Integration** - To evaluate the feasibility and potential benefits of integrating robotic systems with the EPICS framework, aiming to enhance automation and precision in sample handling processes.

1.3 Outline

This thesis is organized into several chapters, each detailing a specific aspect of the research, development, and findings related to the autonomous robotic system for sample switching at the ESS. The structure of the thesis is as follows:

- **Chapter 1: Introduction** - Introduces the context, background and objectives, emphasizing the need for an autonomous robotic system at ESS to improve sample switching processes. Outlines the project's methodology and goals.
- **Chapter 2: Theory** - Covers theoretical foundations, including positioning and movement, kinematics, and homogeneous transforms. Introduces ROS, EPICS, and key concepts in robotic perception.
- **Chapter 3: System Design** - Details the architecture and components of the robotic system. Describes hardware (Stäubli TX60, Intel RealSense camera, 3D-printed handles) and software (movement, perception, system control nodes and EPICS integration).
- **Chapter 4: Developed System Architecture** - Explains the system architecture from user input to robot actuation. Describes inputting sample and task information, FSM operation for task execution, and EPICS integration.
- **Chapter 5: Results** - Presents findings from subsystem and system testing. Includes performance metrics for movement and perception systems, hand-eye calibration accuracy, and overall system results.
- **Chapter 6: Discussion** - Analyses key findings, discussing implications, limitations, and challenges. Reflects on design choices, sensor performance, and system integration. Revisits research questions and goals. Suggests potential enhancements and expansions of the thesis.
- **Chapter 7: Conclusions** - Summarizes contributions, highlighting the feasibility and benefits of the developed system for sample switching at ESS. Emphasizes advancements in robotic automation, system design, and integration.

1.4 Contributions

The work presented in this thesis was a collaborative effort, with most of the tasks being performed jointly by the authors. However, much of the Hand-Eye calibration work can be attributed to Anton Håkansson and most of the EPICS integration can be attributed to Anthon Andersson.

Generative AI tools were used throughout the work. ChatGPT was used to assist in troubleshooting and to suggest improvements to code and written text [OpenAI, n.d.]. GitHub Copilot was used during the development for auto-completion and generation of boilerplate code [GitHub, n.d.].

1.5 Abbreviations

Table 1.1 List of abbreviations.

Abbreviation	Meaning
CV	Computer Vision
DB	Database
EPICS	Experimental Physics and Industrial Control System
ERIC	European Research Infrastructure Consortium
ESS	European Spallation Source
FOV	Field of view
FSM	Finite State Machine
ICS	Integrated Control System
ID	Identification
I/O	Input / Output
IOC	Input / Output Controller
PV	Process Variable
IR	Infrared
JSON	JavaScript Object Notation
KDE	Kernel Density Estimation
MGPI	Move Group Python Interface
OPI	Operator Interface
PC	Personal computer
PnP	Perspective-n-Point
PV	Process Variable
RANSAC	Random Sample Consensus
RGB	Red Green Blue
ROS	Robot Operating System
SQL	Structured Query Language
STL	Stereolithography
UI	User Interface
UML	Unified Modeling Language
URDF	Unified Robotics Description Format

2

Theory

In this chapter, the theoretical background necessary to understand the development and operation of the computer vision-based robotic system for flexible sample switching at the European Spallation Source (ESS) will be presented. This includes the fundamental concepts of positioning and movement, the workings of the Robot Operating System (ROS), the principles behind the Experimental Physics and Industrial Control System (EPICS), and important concepts relating to robotic perception.

2.1 Positioning and Movement

Poses and Transforms

Understanding poses and transforms is a key aspect in both the movement and perception part of this work. The terminology used in this thesis is based on [Lynch and Park, 2017]. Although poses and transforms are mathematically described in similar ways, their differences can be summarized as follows:

- **Pose** - Is the position and orientation of an object in space with reference to a coordinate system.
- **Transform** - Is the transformation between poses or coordinate frames.

There are several conventions and ways to represent poses and transforms that have different properties and use cases. The following has been used in this work:

Positions and Translations Positions and translations are described with a vector of the movement $P = [x, y, z]^T$ with reference to a right-handed coordinate system.

Orientations and Rotations

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.1}$$

Orientations and rotations are described with 3×3 rotation matrices, or parameterized using Euler angles $\phi = [r, p, y]^T$, Rodrigues vectors $r = [a, b, c]^T$ or quaternions $q = [x, y, z, w]^T$.

Homogeneous Transforms The homogeneous transform is a compact representation of a translation and rotation as a single 4×4 matrix.

$$ {}^{A}T_{B} = \begin{bmatrix} {}^{A}R_{B} & {}^{A}P_{B} \\ 0 & 1 \end{bmatrix} \tag{2.2} $$

In Equation (2.2) the transformation from frame B to frame A can be seen.

$$ {}^{A}P = {}^{A}T_{B}\,{}^{B}P = {}^{A}T_{B}\,{}^{B}T_{C}\,{}^{C}P = {}^{A}T_{C}\,{}^{C}P \tag{2.3} $$

These transformations can be compounded as seen in Equation (2.3).

Kinematics

Forward Kinematics Forward kinematics is the problem of finding a pose from a set of joint variables, for example what the current end effector pose is [Popovic and Bowers, 2019].

Inverse Kinematics Inverse kinematics is the problem of finding joint variables for a pose. For example, how to configure the joints to get the end effector to a given pose [Popovic and Bowers, 2019].

2.2 ROS

The Robot Operating System (ROS) is an open-source, meta-operating system for robot applications that provides a set of open-source libraries and tools [*Introduction - ROS Wiki* 2018]. As it was widely used throughout the project, this section gives an overview to some core concepts and tools.

Concepts

Nodes A ROS Node is a runtime process. It is common for a system to have multiple nodes handling different functionality such as sensor management, motor control and user input [*Nodes - ROS Wiki* 2018].

Messages Nodes communicate with each other using messages with a publish / subscribe model. Publishers publish their messages to topics that subscribers can listen to. There are existing standard messages one can use, or it is possible to create custom messages [*Messages - ROS Wiki* 2016].

Services and Actions Services are built on top of the message model and allows nodes to ask other nodes to do work for them through request / reply message passing [*Services - ROS Wiki* 2019]. Actions extends the idea of services by offering monitoring and preemption of tasks [Marder-Eppstein et al., 2024].

Tools

Roslaunch To easily start multiple nodes and set parameters the roslaunch package can be used. Information regarding the nodes to start and any parameters are specified in a .launch file [Conley et al., 2019]. When the file is launched, it runs the commands inside it. This is very useful for starting complex systems and keeping track of parameters.

RViz RViz is a 3D-visualization tool that can visualize the environment as the robot perceives it. It can for example show how the coordinate frames are positioned, how the robot links are configured, point clouds from depth cameras and much more [Hershberger et al., 2018]. This is useful both for operators and developers.

MoveIt MoveIt is a motion planning, kinematics and control tool for controlling robots together with ROS [*MoveIt* n.d.]. By configuring it with a model of the robot, adding environmental obstacles and manipulable objects, it is able to plan and execute trajectories with collision avoidance.

tf2 Tf2 is a ROS transform library that tracks coordinate frames and allows for transformations between any two connected frames. The overall structure is a tree, meaning each frame can have multiple children but only one parent. The tracked transformations can either be static, like the transformation from the world to the base of the robot arm, or dynamic like the transformation between two joints in the robot arm [Foote et al., 2019].

2.3 EPICS

EPICS stands for the Experimental Physics and Industrial Control System, which is an open-source software that provides software infrastructure and tools for large distributed systems, particularly large science experiments like the European Spallation Source [*About EPICS* n.d.]. EPICS allows for high-bandwidth hard and soft real time applications across thousands of computers. This makes it possible to communicate with these computers via channel / PV access, e.g., from a central control room. EPICS normally interacts with real-world I/O and local control tasks.

2.4 Perception

Robotic perception is the capability of robots to understand and interact with their environment using sensory data, environment modelling, and machine learning algorithms. It involves processing data from various sensors, which can be onboard the robot or external, to create a detailed representation of the surroundings. This enables robots to perform tasks such as obstacle detection, object recognition, and navigation. [Premebida et al., 2018]

Camera model To map coordinates in the image, (u, v), to positions in the world, (X, Y, Z), as shown in Equation (2.4), the camera's intrinsic matrix K and extrinsic matrix T must be known [Hemayed, 2003]. The scale factor s marks the depth of the 3D point.

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = K\,T \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.4}$$

Intrinsic parameters - The K matrix, describing the camera's intrinsic parameters, is used to project points from the camera coordinate frame to pixel coordinates. In this thesis the K matrix is obtained from the ROS `CameraInfo` message [*CameraInfo.msg* 2022]. The K matrix is defined as follows in Equation (2.5), where f_x and f_y are the focal lengths, and (c_x, c_y) are the coordinates of the principal point.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Extrinsic parameters - To convert the positions in the camera's coordinate frame to other coordinate frames, the camera's extrinsic matrix needs to be known. This matrix, which represents the transform between the camera frame and the other frame, is obtained through extrinsic calibration, such as hand-eye calibration.

ArUco Markers ArUco markers are square, black and white patterns designed for easy detection in images. Two such markers are shown in Figure 2.1 have unique IDs for each marker pattern, allowing for distinct identification of different markers. The core of their functionality lies in the high contrast design, which makes the detection of the key features easier and more robust. [*Detection of ArUco Markers* n.d.]

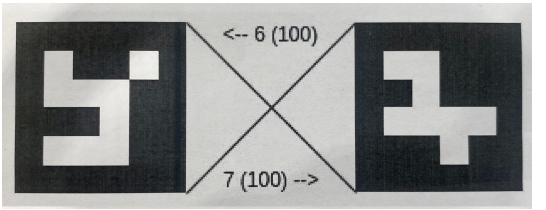


Figure 2.1 Example of ArUco tags with ID 6 on the left, and ID 7 on the right.

The ArUco library, integrated within OpenCV, provides tools for generating, detecting, and estimating the pose of these markers. The ArUco module in OpenCV offers functionalities to detect ArUco markers, estimate their pose, and manage marker dictionaries [*Detection of ArUco Markers* n.d.].

2.5 System Controller

Finite State Machine Finite State Machines (FSMs) consists of a unique set of states. Depending on input and current state, it transitions between these predefined states [Wilson and Mantooth, 2013].

3

System Design

This chapter details the system architecture of the sample switching robot, divided into two primary sections: Hardware and Software. Each section is divided further into three chapters: Movement, Perception and System Control, mirroring the three parts of the autonomous system.

3.1 Preface

At the start of the project, there were pre-existing components in which no development work was involved:

1. **Stäubli TX60 Industrial Arm with Pneumatic Gripper**
2. **CS8C Robot Controller**
3. **Safety cage for the Stäubli TX60**

3.2 Hardware

Stäubli TX60 The robot arm used in the project was the Stäubli TX60, a 6-axis industrial robotic arm, it can be seen in Figure 3.1 and has the technical specifications described in Table 3.1. It was developed by Stäubli, a Swiss company that provides industrial and mechatronic solutions [*Stäubli* n.d.]. Attached to the robot is a pneumatic gripper, capable of transitioning between its two states: open or closed.

CS8C Controller The CS8C controller controls movement of the robot through commands. It also features a user-operated pendant, connected by a cable, for manual control.

Figure 3.1 The Stäubli TX60 industrial robot with pneumatic gripper.

Table 3.1 Technical specifications from datasheet [Stäubli, 2007].

Characteristics	Model
Maximum Load	9 kg
Nominal Load	3.5 kg
Reach (axis 1 - 6)	670 mm
Degrees of Freedom	6
Repeatability (ISO 9283)	$\pm$0.02 mm
Motion Range - Axis 1	$\pm$180°
Motion Range - Axis 2	$\pm$127.5°
Motion Range - Axis 3	$\pm$142.5°
Motion Range - Axis 4	$\pm$270°
Motion Range - Axis 5	$+133.5° - 122.5°$
Motion Range - Axis 6	$\pm$270°

Robot Cage The robot is installed within a cage featuring an aluminium frame and clear acrylic glass sides, seen in Figure 3.2. The cage box outside dimensions measure 1250 (l) $\times$ 1000 (w) $\times$ 1354 (h) mm. It includes a sliding acrylic glass door integrated with the robot controller, restricting operations to manual mode when the door is open.

Sample Holder and Rack To enhance the robot's ability in handling samples of varying sizes and shapes, a sample holder has been engineered to serve as an inter-

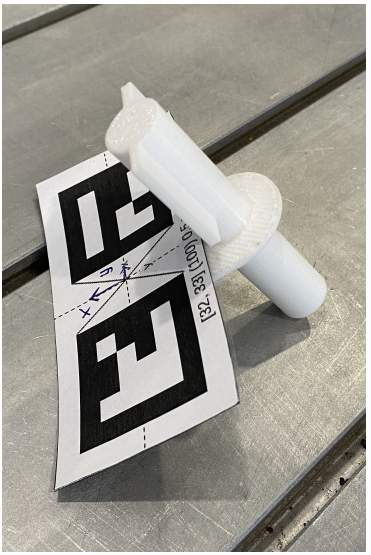
Figure 3.2 Stäubli TX60 in the safety cage.

face between the robot's gripper and the samples. The prototype holder, manufactured through 3D-printing, can be seen in Figure 3.3a. Key features of the handles include:

- **Gripper interface** - The top of the sample features a cylindrical section with two ears on either side. The cylindrical section matches the cutout in the gripper and the ears fit between the gripper's fingers ensuring that smaller misalignments are corrected when the handle is picked up.
- **Base** - To allow for the handle to sit on top of the rack's ports, it has a circular base extending from the centre of the cylinder.
- **Marker platform** - For the perception system to detect the handles, markers are required. Therefore, the base extends on one side into a platform where markers can be added. The 2×1 ArUco grids used can be seen attached to the platform in Figure 3.3a.
- **Sample rod** - Below the base, a rod extends where samples are intended to be attached to the bottom. This is included to give clearance between the rack and the samples when the handle is held in a lifted position in the rack during the pick and place procedures.
- **(Foot)** - Some unique handles, such as the target later used in the hand-eye calibration, include a wide foot at the bottom of the sample rod. When neces-

sary, the foot is included for the attachment of large markers at the bottom of the handle, or to allow for placement of the handle outside of rack ports.

The sample rack, for the placement of handles, has been manufactured out of wood. It can be seen, painted in a green colour, in Figure 3.3b. It has ports where handles with samples can be moved into from the side and placed. An ArUco grid is attached to one corner to allow for detection using the perception system and as more area is available, a 2×2 grid is used.



(a) Sample handle with two ArUco tags.

(b) Sample rack in green, featuring the sample handle in (a). The rack's marker can be seen on the near corner.

Figure 3.3 Images of the sample handle and rack.

Cameras At the start of the thesis, two monochromatic network cameras where provided. These were found to be unsuitable for the task because of the following reasons:

- Sensor being monochromatic.
- Narrow field of view for available lenses.
- Unreliable network connection and complicated configuration required to avoid flooding the network with traffic.
- Low frame rate.

Therefore, it was decided to procure an Intel RealSense D435if depth camera, seen in Figure 3.4. This was decided on for the following reasons:

- Being a stereo camera, only one is required to get depth information. The picked models range of 0.2 to 3.0 m aligns with the distances in this application [*Intel® RealSense™ Product Family D400 Series* 2023].
- The field of view of 69×42 degs. is wide enough to see the majority of the cage's interior when mounted inside [*Intel® RealSense™ Product Family D400 Series* 2023].
- Simple to interface with through a USB connection.
- Manufacturer support for ROS with an open source ROS1 wrapper available.
- Price within with the project's allocated resources.

The depth information that the sensor provides in addition to RGB-data is calculated by having two image sensors identifying matching points in the respective frame and calculating the shift between them. To improve the performance, it has an IR projector that projects an invisible pattern that the image sensors can easily identify and compare. The "f" in the product name indicates that the model used is equipped with an IR-pass filter. This filter mitigates false depth data caused by repetitive patterns in the scene. It is specified to provide depth data with an absolute error of $\pm 2\%$, spatial noise of $\leq 2\%$ and temporal noise of $\leq 1\%$ [*Intel® RealSense™ Product Family D400 Series* 2023].

Mounting Bracket To mount the camera to the robot cage. A mounting bracket was designed and manufactured through 3D-printing. In Figure 3.4 it can be seen how it connects to the ball joint from the camera's complementary tripod, allowing for adjustments of the camera orientation. It is connected to a preexisting bracket in the cage with two screws. No permanent alterations had to be made to the cage.

3.3 Software

The software for the project is running in three locations in a layout that can be seen illustrated in Figure 3.5. The three locations are:

1. **ROS system on Linux PC** - The first location is a Linux computer running the ROS system where most of the program logic is.
2. **ROS drivers on CS8C control cabinet** - The second location is the CS8C control cabinet that is running the robot drivers interpreting the ROS commands and moving the robot.

Figure 3.4 The Intel RealSense D435if depth camera with mounting bracket inside the robot cage.

3. **EPICS on PC** - Lastly a third computer can through EPICS provide user input to the ROS system.

As the code running on the cabinet is harder to monitor, time consuming to upload and test, and written in the less known VAL3 language, the decision was made to only add required core functionality to it and have all other program logic in the ROS part. ROS was used as a result of its open-source status, supporting a wide range of robotic platforms. In the initial stages of the project, ROS1 drivers were found for the CS8C controller, which led to the development in this system to be done in ROS1 as opposed to the newer ROS2.

Movement

The movement software is responsible for controlling the robot. The structure of the movement node can be seen in Figure 3.6.

VAL3 ROS driver The CS8C controller utilizes VAL3, a proprietary language specific to Stäubli [*VAL3 language* n.d.]. Due to the limited familiarity and use of VAL3 outside of Stäubli, a decision was made to adopt a more universally recognized approach by leveraging ROS and minimizing the amount of project-specific code written in VAL3. This was accomplished by utilizing the existing `staubli_val3_driver` package that exposes the robot as a ROS server compatible with MoveIt [Martins, 2020]. This configuration allows the robot to be controlled by a ROS system running on a separate computer.

Modbus Gripper Control The ROS drivers for the robotic arm, described in the previous section, lacked functionality for controlling the gripper. This presented a challenge that required the development of a solution for gripper control, compatible with the existing ROS drivers. The solution was to implement a simple VAL3

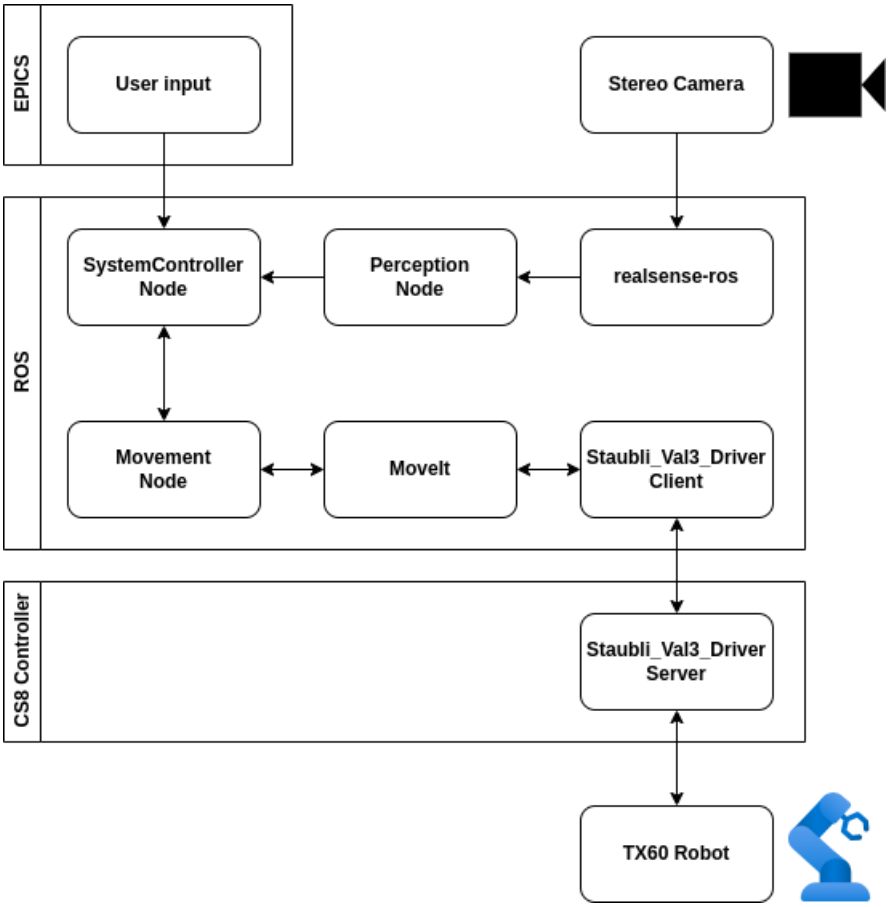


Figure 3.5 System overview with the key components and the location where they are running.

program, add it to the existing `staubli_val3_driver` and deploy the modified driver to the robot. The addition listens to a Modbus port and toggles the gripper to match the port state. Then the port could be written to by the `GripperController` class in the movement node, allowing for gripper control from a separate computer.

MoveIt Configuration In order to control the robot with MoveIt, as required by the VAL3 drivers, a configuration was necessary. As none was available, a new configuration was needed to be created. To create a configuration, a URDF file describing the robot is required. An existing URDF describing the robot arm was available in the package `staubli_experimental`[vd. Hoorn, 2018], however, the file did not contain any information about this robot's end effector. Therefore, the necessary geometries were modelled and a URDF file for the end effector was created.

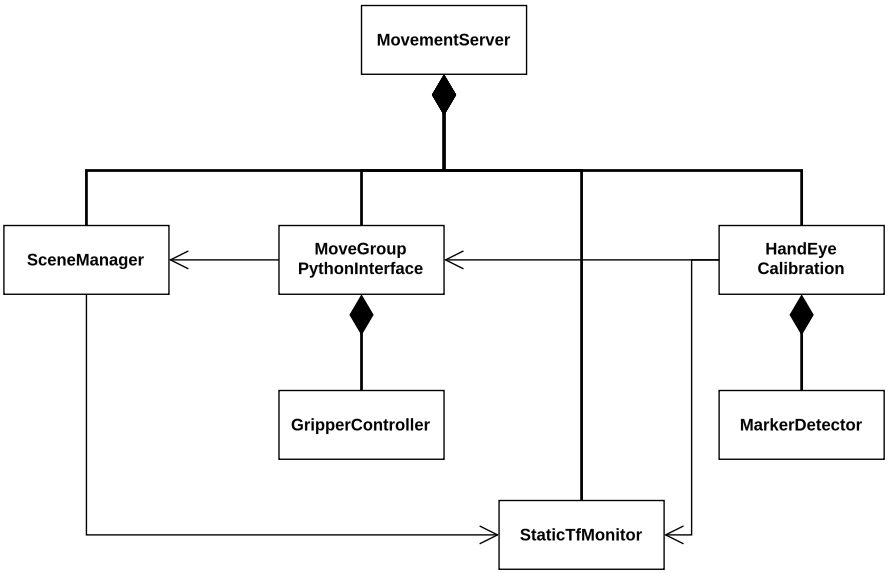


Figure 3.6 Simplified UML diagram of the Movement Node.

Now with one file describing the arm and one describing the end effector, a file describing the entire robot could be created.

With the entire robot described, the graphical user interface "MoveIt Setup Assistant" [*MoveIt Setup Assistant* n.d.] could be used to create the MoveIt configuration. With this tool, the ROS package `tx60_with_eef_moveit_config` was generated and after necessary manual file alterations to accommodate the VAL3 drivers used, it was possible to control the robot through MoveIt. To facilitate an easy start-up process for users, a launch file containing information specific to this individual robot and thesis was created. When launched it starts MoveIt, the VAL3 drivers and RViz, and connects to the server running on the robot.

Move Group Python Interface The `MoveGroupPythonInterface` (MGPI) class is a Python interface for controlling the movement of the Stäubli TX60 robot. This class provides control mechanisms, including setting joint states, setting the end-effector pose, and gripper operations. Additionally, it has capabilities to record robot states and offers methods for reporting on the robot's status and operational conditions. To allow for safe movement when testing new features or after a new calibration, it is possible for the user to preview the planned paths in RViz and approve them before any commands are sent to the robot. This was developed based on work by [Digumarti, 2023] and a MoveIt tutorial by [Pooley and Lautman, 2013].

Scene Manager The `SceneManager` class was developed as a wrapper for MoveIt's `PlanningSceneInterface` class to streamline the management of ob-

jects in MoveIt's planning scene. This class simplifies the addition, deletion, and modification of objects within the scene. It is designed to accommodate a dynamic array of objects without imposing constraints on the scene configuration, enhancing the adaptability of the environment to various robotic tasks and simulations.

Scene changes can be done via method calls from classes within the same node or through service requests from classes in other nodes. The scene manager also allows other nodes to retrieve a list of the currently empty rack ports which allows for safe placement of samples without the user specifying a specific location.

Movement Server The `MovementServer` is the top-level class in the Movement Node and receives commands from the `MovementClient` in the system control node. It ensures successful actuation of the robot by verifying the commands and executing them by calling different MGPI methods. It exposes four different action servers:

- `pickup_sample` - Partial implementation of MoveIt's Pickup action capable of picking up sample handles from racks. The steps in the pickup can be seen in Figure 3.7.
- `place_sample` - Partial implementation of MoveIt's Place action capable of placing sample handles in racks. The steps in the placement can be seen in Figure 3.7. Notably, if the pick fails after the sample has been grasped, it will try to place the sample and move away to make the failure easier to manage.
- `move_to` - A custom action that moves the end effector to the pose specified in the action goal.
- `go_home` - A custom action that moves the robot to the home position, where all joint angles are zero and it is completely vertical. This is intended to move the robot out of the way for the perception system.

Partially implementing MoveIt actions was decided on as MoveIt's generic action definitions allow for additional complexity and features to be added later, even if they are not currently used by the system. Furthermore, this approach ensures the node uses a standard interface, facilitating its integration with other ROS software.

Perception

The structure of the perception node can be seen in Figure 3.8.

RealSense ROS Wrapper To use the RealSense D435if stereo camera with ROS, the existing package `realsense2_camera` was used [*ros-realsense* 2024]. It publishes the sensor data as ROS messages for other nodes to subscribe to. Further, it publishes camera info containing the camera's intrinsic parameters and transformations between the camera's different coordinate frames.

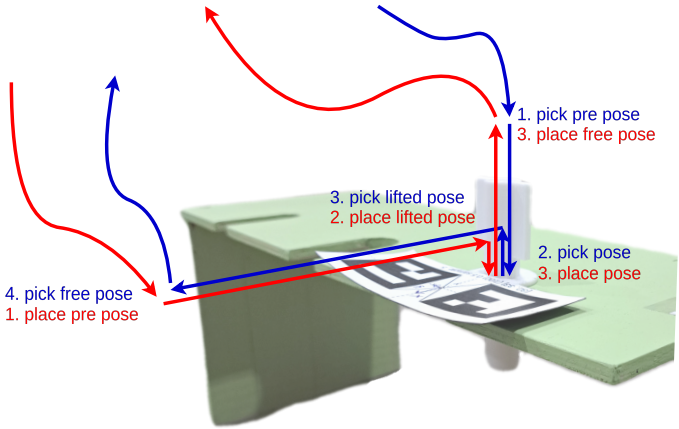


Figure 3.7 Movement during pickup (blue) and placement (red) of samples from the rack. The straight arrows indicate Cartesian movement. The reason for the sideways entry and exit when holding handles is to allow for large samples to be attached.

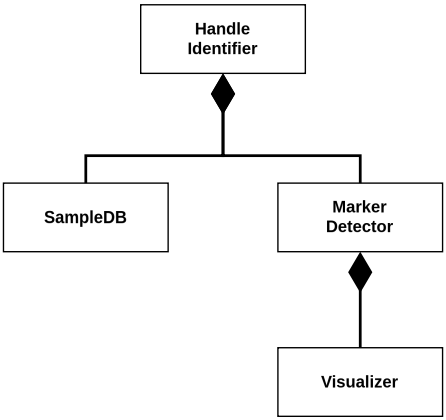


Figure 3.8 Simplified UML diagram of the Perception Node.

Camera Calibration To get adequately accurate and precise measurements from the stereo camera, it was calibrated using the Intel RealSense Viewer, provided in the Intel RealSense SDK 2.0 library [*Intel® RealSense™ SDK 2.0* 2024] in accordance to the instructions provided by [Grunnet-Jepsen et al., n.d.]. The calibration had three main steps:

1. On-chip calibration - Reduces the relative error, or noise.

2. Focal length calibration - Compensates for focal length imbalance between the left and right camera.

3. Ground truth calibration - Corrects for absolute errors in the depth measurements.

Marker Detector The class that was responsible for all operations relating to the detection and pose estimation of ArUco markers was the `MarkerDetector`. It included a dictionary of ArUco IDs and tag sizes, important for identifying and processing different markers. The decision to implement an ArUco-marker based perception system, within the `MarkerDetector` class, was driven by several considerations with the goal of developing a flexible and robust system. Initially, alternative methods using deep learning models like YOLOv8 [Jocher et al., 2023] and DenseFusion [Wang et al., 2019] were considered. However, these approaches faced the following challenges:

1. **High Variability of Sample Types:** The diverse range of possible sample types presented a significant challenge for using deep learning, as it requires extensive pre-training of the specific objects to be detected. Implementing such models would necessitate either limiting the variation of samples or frequently retraining the model for new objects. This approach was deemed inappropriate as a key aspect of the thesis was increased flexibility, which would be hindered by such constraints.

2. **Low Variability within a Sample Type:** Contrary to the point above, for an experiment with a certain sample type, the variability between different samples of that type can be extremely low. This would make it difficult for models to distinguish and identify one sample from another which is important as the right experiment needs to be conducted on the right sample.

3. **Accuracy of Pose Estimation:** To accurately determine the pose of a sample, the position of known key points needs to be determined. This would require the collection and annotation of a large dataset to achieve the required accuracy. This was determined to be infeasible within the scope of this thesis.

4. **Integration and Reliability:** Deep-learning models such as DenseFusion would also suffer from integration issues, as they often have strict version requirements, further complicating the integration into this system.

Instead, by using ArUco markers, the system has the ability to dynamically process a wide array of sample types without constraints on the type of objects. This choice ensures that the perception system remains adaptable and is not limited by the physical characteristics of the samples mounted on the handles, thus maintaining operational flexibility. Further, there are existing methods implemented in OpenCV for object detection and pose estimation of ArUco markers.

The actual markers used for the detection of the sample handles can be seen in Figure 3.9. It was decided to use the smallest predefined dictionary, the `4x4_50` dictionary that has a total of 50 markers. With four markers each reserved for the hand-eye calibration target and the sample rack, 42 markers are left, allowing for the simultaneous tracking of 21 samples with two markers each. This was deemed a sufficient number and allowed for the individual markers to be fairly small while still being detectable by the perception system at a distance.

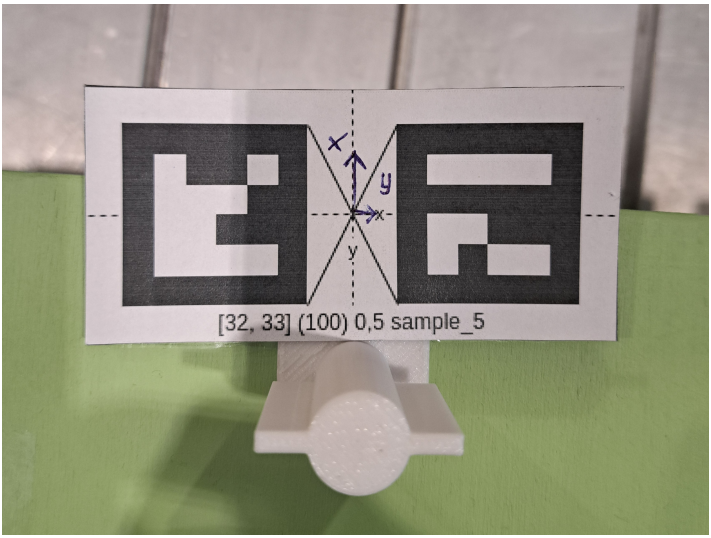


Figure 3.9 Sample handle with marker seen from above. Additional information can be printed on the marker, shown here is the IDs of the individual markers, the marker size, the distance between the markers, the ID of the object, and the coordinate frame of the marker.

The orientation and rotation of the marker was determined separately in the following way.

1. **Identification:** The corners and IDs of the markers in the image frame are identified using OpenCV's `detectMarkers()` method [*ArucoDetector Class Reference* n.d.].

2. **Orientation:** The orientations of the ArUco grids are calculated using Perspective-n-Point (PnP) pose computation using all the corners of the markers in the grid. To better deal with outliers the RANSAC scheme is used [*Perspective-n-Point (PnP) pose computation* n.d.]. As all the corners of the marker in the grid are used, the more markers each grid contains the more accurate and robust the orientation estimation is. This creates a trade-off between accuracy, marker size and the maximum number of objects that can be tracked.

3. **Position:** The positions of the ArUco grids are calculated using Equation (3.1), which uses the depth data, the pixel position for the centre of the grid (u, v), and the camera's intrinsic parameters, shown in Equation (2.5).

$$z = depth\_image[u, v] \tag{3.1a}$$
$$x = (u - c_x) \,/\, f_x \cdot z \tag{3.1b}$$
$$y = (v - c_y) \,/\, f_y \cdot z \tag{3.1c}$$

   Sometimes the depth image included invalid pixels. To avoid this issue, any measurement with a z-value under a predetermined threshold was discarded.

With the robot and camera located within a reflective cage, it was discovered that the system could accidentally detect reflections from the cage sides. This issue was mitigated by removing the disruptive cage sides from the image before the marker identification.

Pose Filtering To get a more robust result, a large number of measurements are gathered each time and then filtered for the most likely pose.

- **Position** - The positions of the measurements are averaged.
- **Orientation** - The orientation is estimated using kernel density estimation (KDE) [Chen, 2017]. To allow for the filter to manage measurements with an extremely low level of variation, a small amount of Gaussian noise is added before estimating. However, this noise is not retained for the returned value.

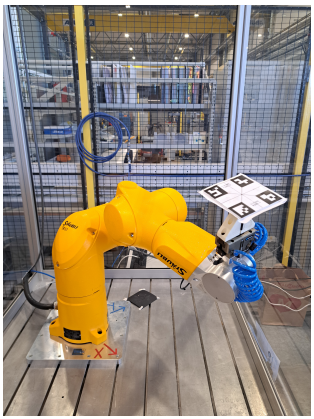
  To further increase the accuracy of the orientation estimations, it was made possible to add a vertical constraint to objects such as the sample rack and samples in the rack. This improved the pose estimation as the PnP is good at determining the yaw angle of a marker when viewed from above but struggles more with the pitch and roll angles. However, for this to work the hand-eye calibration, described in the next section, needs to be completed as the detected pose is made vertical with reference to the robot base frame.

Visualizer `Visualizer` is a class designed to interface with ROS for publishing images with specific overlays onto ROS topics. This streamlines the visualization of results from the `MarkerDetector`.

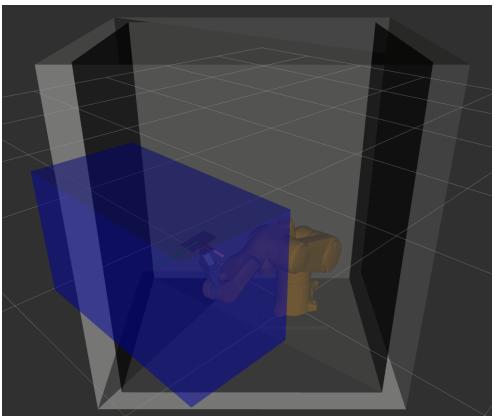
Hand-Eye Calibration To convert measurements, taken in the camera frame, to the world or robot frame, the camera's and the robot's frames need to be in a connected transform tree. To connect the transform trees, the camera's pose with reference to a coordinate frame in the robot's transform tree needs to be known. The `HandEyeCalibration` class calculates and publishes the required transform by solving the eye-to-hand calibration problem. This is done in the following steps:

1. Data gathering: First the robot picks up a calibration target, seen in Figure 3.10a, that the perception system can easily identify the pose of. Then the system generates a random pose within an area in front of the camera, seen in Figure 3.10b, that is facing the approximate camera position. After verifying that the pose is valid and reachable by the robot, it executes the movement, otherwise it generates a new random pose. At each point it attempts to record a data-point that consists of the pair:

- ${}^{c}T_{t}$ - The transformation matrix from the target frame to the camera optical frame. This is obtained by the marker detector.
- ${}^{g}T_{b}$ - The transformation matrix from the robot base frame to the target frame. This is obtained by tf2 from the robot's forward kinematics.



(a) Robot holding the hand-eye calibration target.



(b) The volume that the system generates poses for data gathering within (blue) displayed in RViz.

Figure 3.10 Images of the data gathering process for the hand-eye calibration.

A key aspect in getting a good result is accurate measurements. This is done by taking a large number of measurements at each pose and then filtering them to get the most likely transformation according to the procedure described in the Pose Filtering section.

2. Calculation: After gathering measurements at a predefined number of poses, the hand-eye calibration system uses the list of transformations to calculate the camera pose. The transformations involved can be seen in Figure 3.11. This can be done using the OpenCV function `calibrateHandEye()` that solves Equation (3.2c) to obtain ${}^{b}T_{c}$. The function is able to perform the hand-eye calibration using methods by: [Andreff et al., 1999], [Daniilidis, 1998], [Horaud and Dornaika, 1995],

[Park and Martin, 1994] and [Tsai and Lenz, 1989] [*Camera Calibration and 3D Reconstruction* n.d.].

$$ {}^{g}T_{b}^{(1)}\ {}^{b}T_{c}\ {}^{c}T_{t}^{(1)} = {}^{g}T_{b}^{(2)}\ {}^{b}T_{c}\ {}^{c}T_{t}^{(2)} \tag{3.2a} $$

$$ \underbrace{({}^{g}T_{b}^{(2)})^{-1}\ {}^{g}T_{b}^{(1)}}_{A_i}\ {}^{b}T_{c} = {}^{b}T_{c}\ \underbrace{{}^{c}T_{t}^{(2)}\ ({}^{c}T_{t}^{(1)})^{-1}}_{B_i} \tag{3.2b} $$

$$ A_i X = X B_i \tag{3.2c} $$

$$ \text{where} \quad X = {}^{b}T_{c} $$

An alternative approach was developed that directly computes the transformation ${}^{b}T_{c}^{i}$ for each data-point using Equation (3.3) and then filters the calculated transformations for the most likely one using the method described in the Pose Filtering section. This approach will henceforth be called the direct approach and was the one used for the complete system due to superior performance.

$$ {}^{b}T_{c} = ({}^{c}T_{b})^{-1} = ({}^{c}T_{t}\ {}^{g}T_{b})^{-1} \tag{3.3} $$

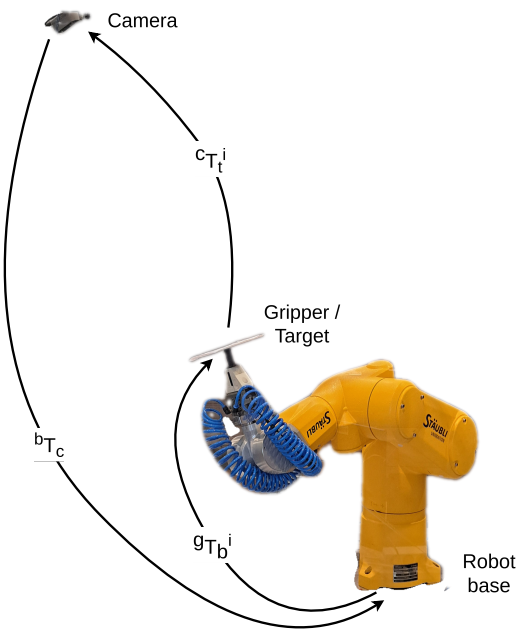


Figure 3.11 The transformations between the robot base, gripper / target, and camera.

3. Publish transform: After the transform ${}^{b}T_{c}$ has been computed, a transform that can be published to tf2 needs to be obtained. Since the camera optical frame and

the robot base frame already have parents those cannot be used directly. An intuitive choice it to calculate the transformation from the world frame to the camera link frame. This is done by the following steps:

1. Transform the pose of the origin of the camera link frame to the camera optical frame.
2. Publishing a fake robot base frame from the camera optical frame using the calculated transform. Then the pose can be transformed to the fake base link frame. This circumvents the problem of the robot base frame already having a parent.
3. Now the pose can be transformed from the real robot base frame to the world frame.
4. Finally, the pose, now in the world frame, can be used to publish a transform from the world frame to the camera link frame.

With the camera and robot frame connected, creating the transform tree seen in Figure 3.12, it is now possible to convert the pose of objects observed by the camera to any other frame in the tree. As the hand-eye calibration only needs to be completed once, the `HandEyeCalibration` class has capability for saving and republishing transforms, resulting in a fast and predictable system once a calibration is complete.

In addition to publishing the transform to the camera, this class is responsible for spawning the objects in the scene that might have moved between executions but are static when the robot is running, such as the sample rack. Although the class has a `MarkerDetector` and its task is related to perception, it is located in the movement node as it is an integral part of the startup of that node, and it sends movement commands to the robot through the `MGPI`. Notably, the `MarkerDetector` is only instantiated when new data is recorded. This design allows the class to perform its other duties even without a connected camera.

Handle Identifier The `HandleIdentifier` is the top-level class in the perception node. It is responsible for the object detection and pose estimation of all samples in the scene. It is constantly running, estimating the pose of all known samples, and publishing a custom message of their object ID and pose. It uses the `MarkerDetectors` methods but converts the detected pose to one that matches the handle frame seen in Figure 3.13. This makes it easy for other nodes to use the perception results directly without having to deal with frame conversions and matching markers to objects.

System Control

The system control node is responsible for processing user inputs, such as sample information and tasks. Then it is responsible for managing the execution of the

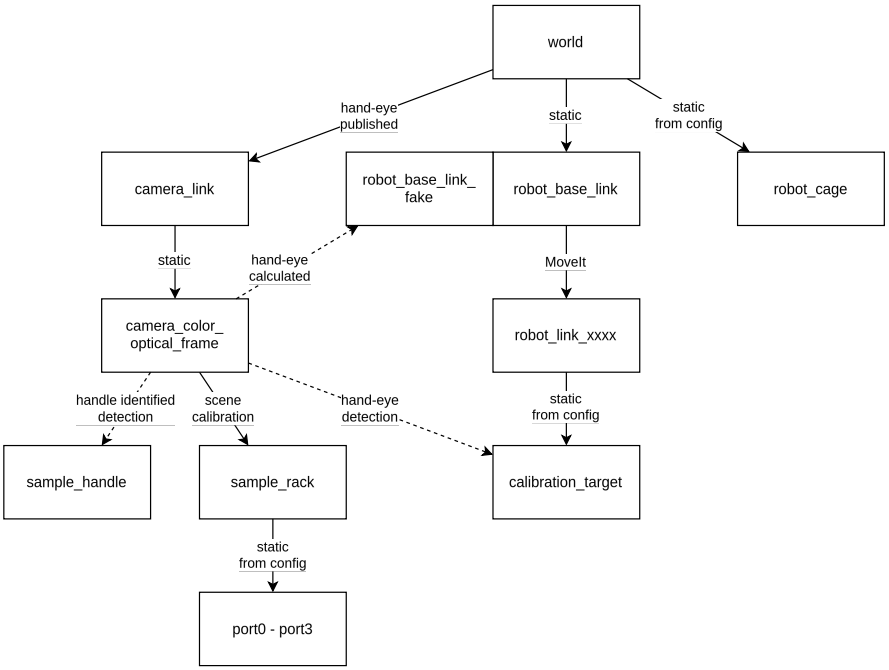


Figure 3.12 Diagram showing the system's transform tree. Solid lines represent published transforms and dashed lines represent observations or calculations.

tasks by sequentially requesting the movement node to execute each step. A UML overview of the node is shown in Figure 3.14.

Sample Database The `SampleDB` database plays an important role in the operation of the perception and system control node by managing information about each sample. To ensure concurrency safety, the database is implemented with SQL. The *Data* field is originally a list of integers. However, due to the nature of SQL, the list is encoded into a JSON string for storage in the `SampleDB`. A detailed description of each field in the database is in Table 3.2.

Sample Manager To coordinate operations related to samples between the perception node and `SampleDB`, the `SampleManager` was created. An overview of the class can be seen in Figure 3.15. It has several key features that is integral to the autonomous system:

1. **Message Listening and Object Spawning** - `SampleManager` continuously listens to the `/IdAndPoseList` message via a callback. This message contains the IDs and poses of objects detected by the perception system. The `SampleManager` checks if any of the detected objects exist as entries in `SampleDB`. If an object is found, it is spawned in the MoveIt

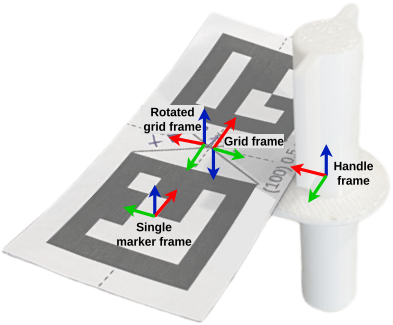


Figure 3.13 A sample handle with a marker showing the different coordinate frames.

Table 3.2 Fields in the `SampleDB` Database.

Field	Type	Description
Object ID	String	A unique identifier for each object.
STL Path	String	Relative file path to the STL file of the object.
Rows	Integer	Number of rows in the ArUco grid.
Cols	Integer	Number of columns in the ArUco grid.
Data	String	Unique ArUco IDs stored in the grid.
Marker Size	Float	Size of the ArUco tags in the grid, measured in meters.
Intermarker Quota	Float	Ratio of distance between consecutive markers.

`PlanningScene`, ensuring proper collision avoidance.

2. **Object Blacklist** - The `SampleManager` has an object blacklist, which contains the IDs of objects that should not be tracked. This is for the situations when the robot is interacting with a sample, as it will then publish a more accurate pose for the sample that should not be overwritten.

3. **Sample Database Updates** - `SampleManager` has a `SampleDB` and exposes services for adding sample information to it.

Movement Client The `MovementClient` sends actuation commands from the system control node to the movement node. It has three main functions:

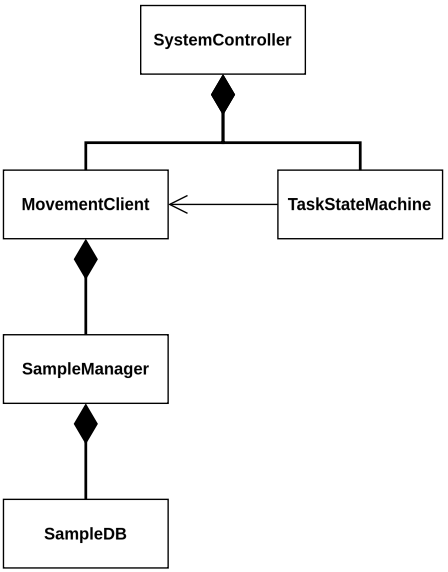


Figure 3.14 Simplified UML diagram of the System Controller Node.

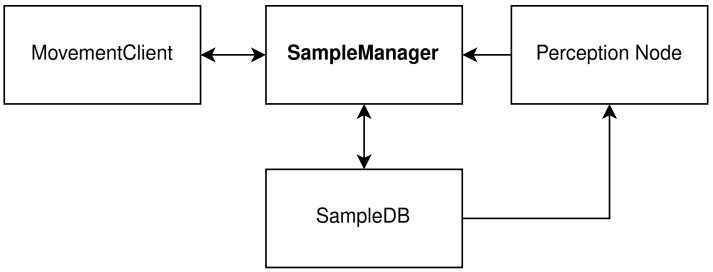


Figure 3.15 Overview of SampleManager interactions.

1. `pickup_sample_handle` - Sends a request to the `MovementServer` to pick up a sample handle based on its object ID. If this request succeeds, then the object is blacklisted in the `SampleManager`.

2. `place_sample_handle` - Sends a request to the `MovementServer` to place the sample handle in the rack. If the place fails when no specific pose was provided, it will reattempt to place the sample handle in another free rack port.

3. `go_home` - Sends a request to `MovementClient` to set all joint states to zero.

Task State Machine The system controller utilizes a finite state machine (FSM) architecture to execute tasks. The FSM framework is implemented using the

SMACH library designed for ROS applications [Bohren, 2018]. The states and actions connected to it are discussed more in-depth in Chapter 4.

System Controller The `SystemController` class is the top level class in the system controller node. It is responsible for starting the node as well as exposing a service that allows other nodes to add task to the execution queue that the `TaskStateMachine` then executes.

EPICS

In this thesis, the integration with EPICS (Experimental Physics and Industrial Control System) takes a unique approach. Typically, integration involves creating a soft input/output controller (IOC) along with an EPICS database to represent all process variables (PVs). However, this implementation is different. Instead of the usual method, a server application is developed to control and manage all PVs. This approach leverages the P4P (PVAccess for Python) library, which is a Python wrapper around the PVAccess protocol network and is a part of the EPICS framework [Davidsaver, 2023]. All necessary data for PV creation and data exchange with the system control is loaded from a JSON file. Consequently, bidirectional information exchange between the server and the client with the ROS software can be established. The server is controlled by user using Phoebus [*Phoebus 1.0 Documentation* n.d.], which is an operator interface commonly used in the EPICS community.

ROS-Client The `Client` subscribes to two process variables, and listens for any updates to the variables. It translates EPICS datatypes into ROS messages.

Server The `Server` sets up handlers to manage updates to the process variables, and initializes these variables based on the JSON configuration. It then runs indefinitely.

OPI The Operator Interface (OPI) is a Graphical User Interface (GUI) built using Phoebus. Users can utilize this interface to update sample information, select which sample and task information should be sent to the robot, and execute the tasks.

4

Developed System Architecture

This chapter provides a walkthrough of the entire workflow—from initial input to final actuation, and through to the representation of information within the system. The goal is to provide a clear understanding of the system's architecture and operation. By examining each stage in detail, this chapter showcases the project's complexity and technical requirements. A high-level overview can be seen in Figure 4.1. Apart from the camera's sampling frequency of 30 Hz, the developed system is callback based. This means that tasks are executed as soon as they become available, otherwise the system is waiting and only updating its scene when new images arrive.

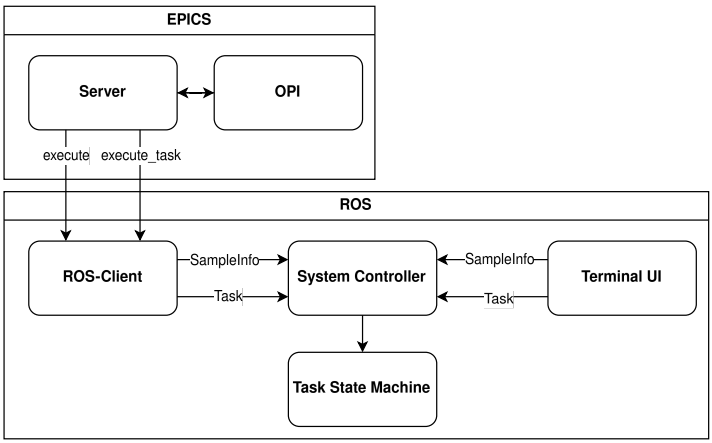


Figure 4.1 High level overview of control interactions.

User Input

The process begins with the user's initial input, which serves as the foundation for operations within the program. At this stage, the user is required to provide information regarding which samples will be in the scene. Then the user can provide tasks for the system to execute with the samples. These inputs are entered into the program either via a user interface on the ROS computer or through EPICS.

Sample Information The sample information needs to be provided to the system so it knows which samples it can expect in the scene, which markers they are identified by, and what the sample geometries are. This is import for collision avoidance and pose estimation. The required information is:

- **Object ID** - To manage and identify samples throughout the process, the user provides a unique identifier for each object. This name is used to track the object within the system and to reference it in logs and output data.
- **ArUco Information** - Each object is associated with a target for the vision system. This is typically a row of two markers but can be any matrix of markers.
- **STL Path** - The user must specify the path to an STL file corresponding to the object. This file is essential for rendering the object within the scene, which is required for accurate collision avoidance.

Task Information In addition to sample information, details regarding the tasks to be executed by the system can be provided. This consist of the following segments.

- **Object ID** - The object ID is the same unique identifier as in the task information, allowing the system to connect tasks with the samples that they should be conducted on.
- **Actions[]** - The overall task is broken down into a list of smaller actions that can be combined in any way the user likes. For example, an experiment where the sample is supposed to be held in the beam for a set amount of time could be a task with the following list of actions: `pick`, `move_to`, `wait`, and `place`. Each action requires information of its type, and the following additional information can or has to be provided:
  - `pick`: Optional to provide a pick pose, if none is provided the system locates the sample using the perception system.
  - `place`: Optional to provide a place pose, if none is provided the system identifies a free rack port to place the sample at.
  - `move_to`: Required to provide a pose to move to.
  - `wait`: Required to provide the wait duration.

Finite State Machine Operation

The Finite State Machine (FSM) serves as the central control structure, initiating its process once it is created in the system controller node. This section outlines the sequence of states the FSM transitions through during a typical task cycle. The high-level stages of the FSM can be seen in Figure 4.2. The design of this Finite State Machine prioritizes flexibility to manage a diverse range of sample investigations. By structuring the Finite State Machine to handle various task types, the system avoids imposing restrictions on the robot's operational capabilities. This flexibility is important as it allows the robot to adapt to different experimental requirements and tasks without needing reconfiguration.

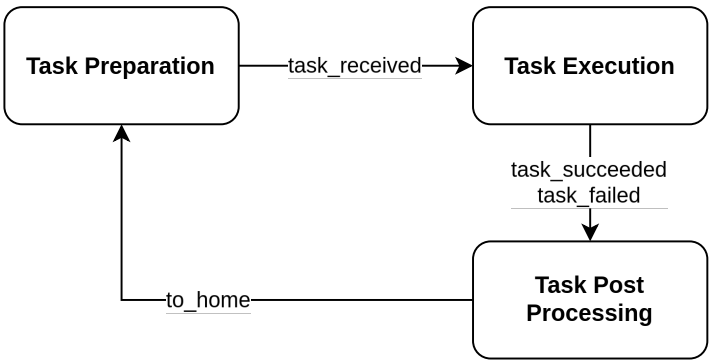


Figure 4.2 FSM high level flow.

1. Task preparation The first stage in the FSM's execution, seen in Figure 4.3, is preparing the system for a new task and fetching one from the task queue. This is done with the following states that execute in sequence:

1. `HomeState` - Requests the robot to move to the joint state where all joint angles are zero. This state is suitable as it prevents the robot from blocking the perception system when the robot is searching for samples.

2. `WaitForTaskState` - This state blocks until it is able to pop a task from the task queue. Once that happens, it outputs the task to the subsequent task execution stage.

2. Task execution The second stage is the task execution. Seen in Figure 4.4, it follows a straightforward fetch-execute cycle. In this cycle, the system repeatedly fetches the next action and then executes it in a dedicated execution state. This process continues while there are actions left to execute or until an action fails. The states involved in this stage are:

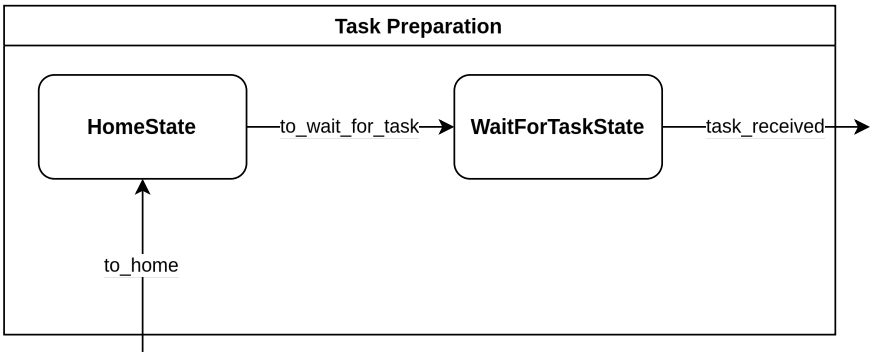


Figure 4.3 FSM preparation stage.

1. **Fetch state**: `GetNextActionState` - This state identifies the next unexecuted action in the task and transitions to the corresponding execute state. If all actions are completed, it transitions to the `TaskSucceededState`.

2. **Execute states**: These states perform their respective actions. If an execution succeeds, the FSM transitions back to `GetNextActionState`; if it fails, the FSM transitions to `ActionFailedState`. The different execute states each correspond to a specific action and are:

   - `PickState` - Sends a `PickupGoal` to the movement node for execution.
   - `MoveToState` - Sends a `MoveToGoal` to the movement node for execution.
   - `WaitState` - Waits for the specified duration.
   - `PlaceState` - Sends a `PlaceGoal` to the movement node for execution.

3. **Failure state**: `ActionFailedState` - If an action fails, the FSM transitions to this state to attempt recovery or transition to the `TaskFailedState`.

The task execution is structured in this way to give the user the flexibility to arrange tasks as they see fit. Additionally, this design allows for easy extension of the system's capabilities. New execution states, implementing new actions, can be seamlessly added to the program.

3. Task post processing. After a task has executed successfully or failed in an unrecoverable way, the FSM enters the final stage of its cycle: the task post processing, which can be seen in Figure 4.5. This part is intended for collecting information and provide feedback to the operator. The post-processing occurs in one of two states:

- `TaskSucceededState` - As everything executed as expected, the system transitions back to `HomeState`.

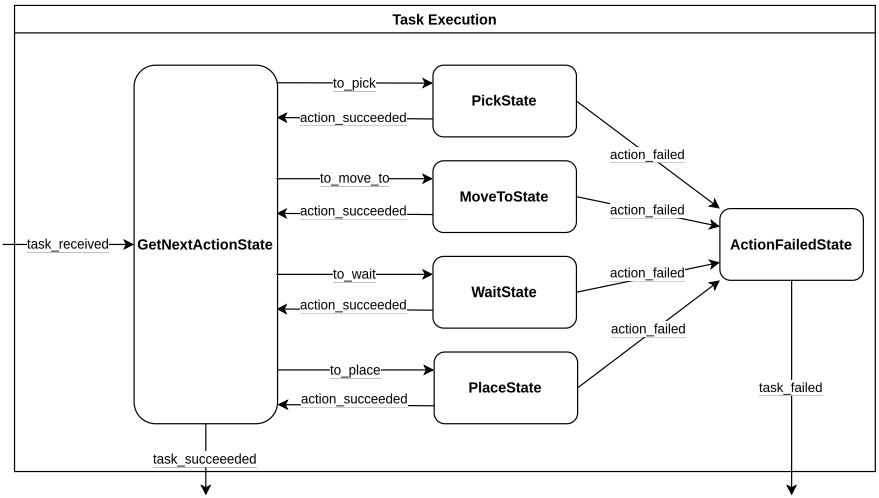


Figure 4.4 FSM execution stage.

- `TaskFailedState` - As an error occurred, this state will stop the system until the user gives permission to proceed to the `HomeState` for the next task.

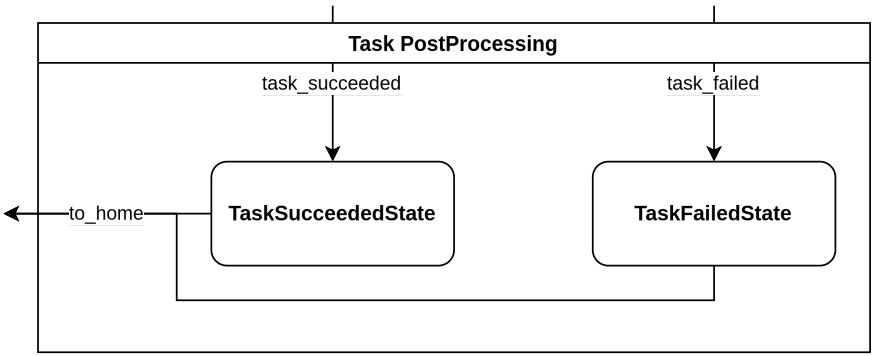


Figure 4.5 FSM post processing stage.

EPICS

To integrate EPICS into the system workflow, a driver was designed to translate EPICS PVs into ROS messages.

Loading of JSON files Upon starting the EPICS server, predefined tasks and samples are loaded from two JSON files. This information populates the process variables.

OPI The OPI, seen in Figure 4.6, serves as the user interface, displaying process variables in dedicated boxes. It features two windows for sample and task information, respectively. Users can switch between tasks and samples using a drop-down menu and execute them with separate buttons. The Add and Remove buttons are not implemented.

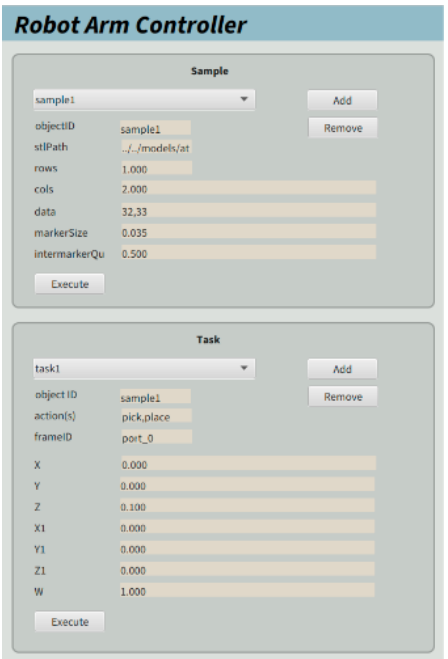


Figure 4.6 The Operator Interface.

Client When execute is clicked in the OPI, process variables are updated. The client listens to `execute` and `execute_task` variables. Callback functions extract and validate the data types of the raw values from these variables. Based on the information sent, a `SampleInfo` or `Task` is created and sent to the system controller server, initiating the robot's operation.

5

Results

This chapter presents the results obtained from the development and testing of the autonomous system for sample switching at ESS. The results are divided into two main sections: subsystems and the complete system. In the subsystems section, data related to pose estimation and hand-eye calibration is detailed, showcasing the performance of individual components. The complete system section presents the full system, including the execution of an operational cycle.

5.1 Subsystems

Movement

The movement system was able to plan and execute both large, complex movements and small, precise adjustments within the environment without colliding with correctly spawned objects. However, the motion could be somewhat jerky, particularly noticeable at the end of the trajectory. The system was also able to actuate the gripper.

Perception

Camera calibration Camera calibration significantly improved the precision of the perception system. Prior to calibration, two different datasets were collected, shown in red and blue in Figure 5.1. These datasets include depth measurements from the camera, taken at known distances. The figure illustrates that the initial absolute errors in distance measurements were substantial and increased with distance. Specifically, absolute errors when averaging 100 measurements were in the range of 1% to 3% and escalated to 4% to 6% at longer distances.

After performing a comprehensive camera calibration, these errors were reduced to an average absolute error of less than 0.5%. A plot of the measurements' deviation and its mean at different distances can be seen in Figure 5.2. The distribution of the measurements at a specific distance can be seen in Figure 5.3.

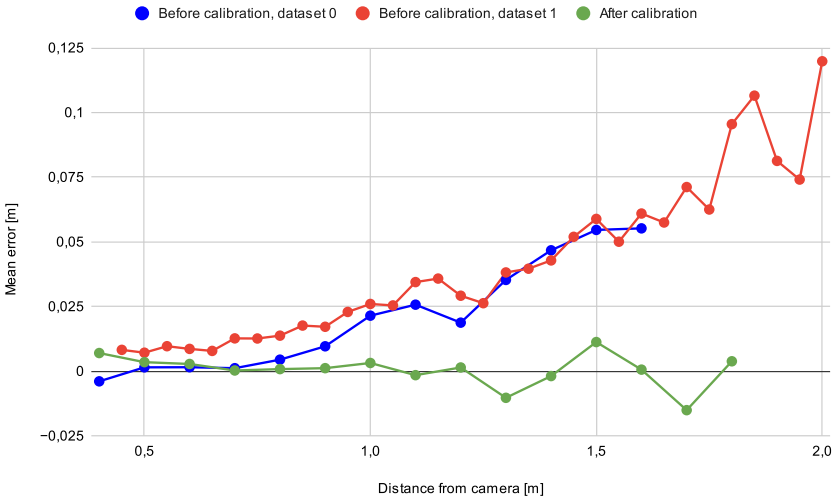


Figure 5.1 Mean errors at different distances, before and after camera calibration. Each data-point is the average of 100 measurements.

Marker detection For the marker detection, key results include the position and orientation estimations. In the previous section the distribution of the position measurement can be seen in Figure 5.3. For the orientation estimation, the measured values, their distribution, and the filtered result can be seen in Table 5.1.

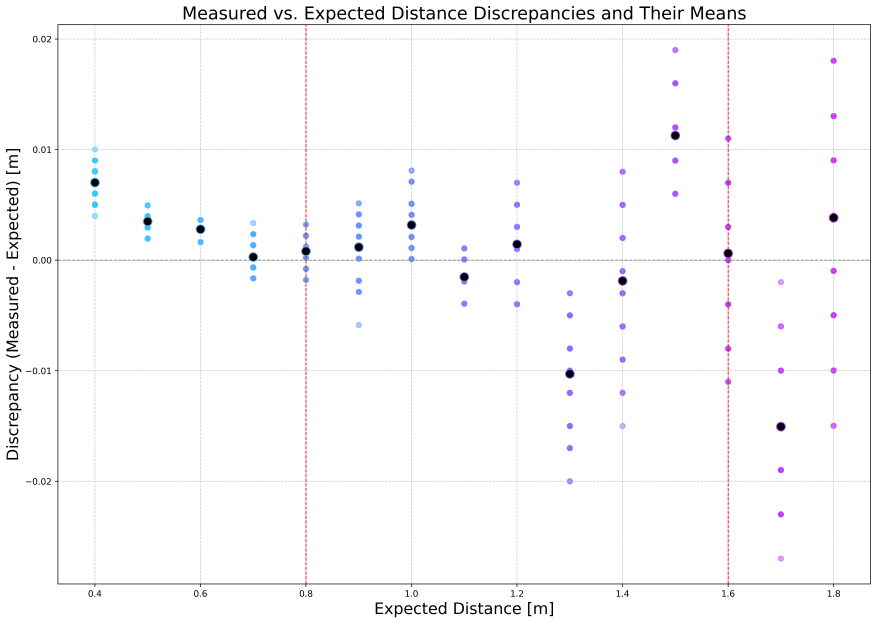
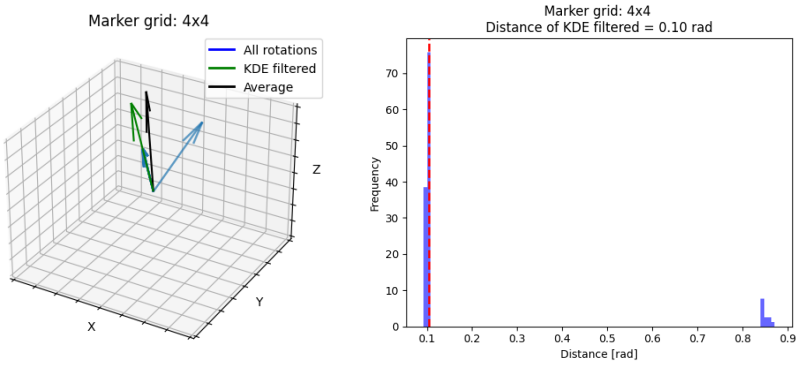


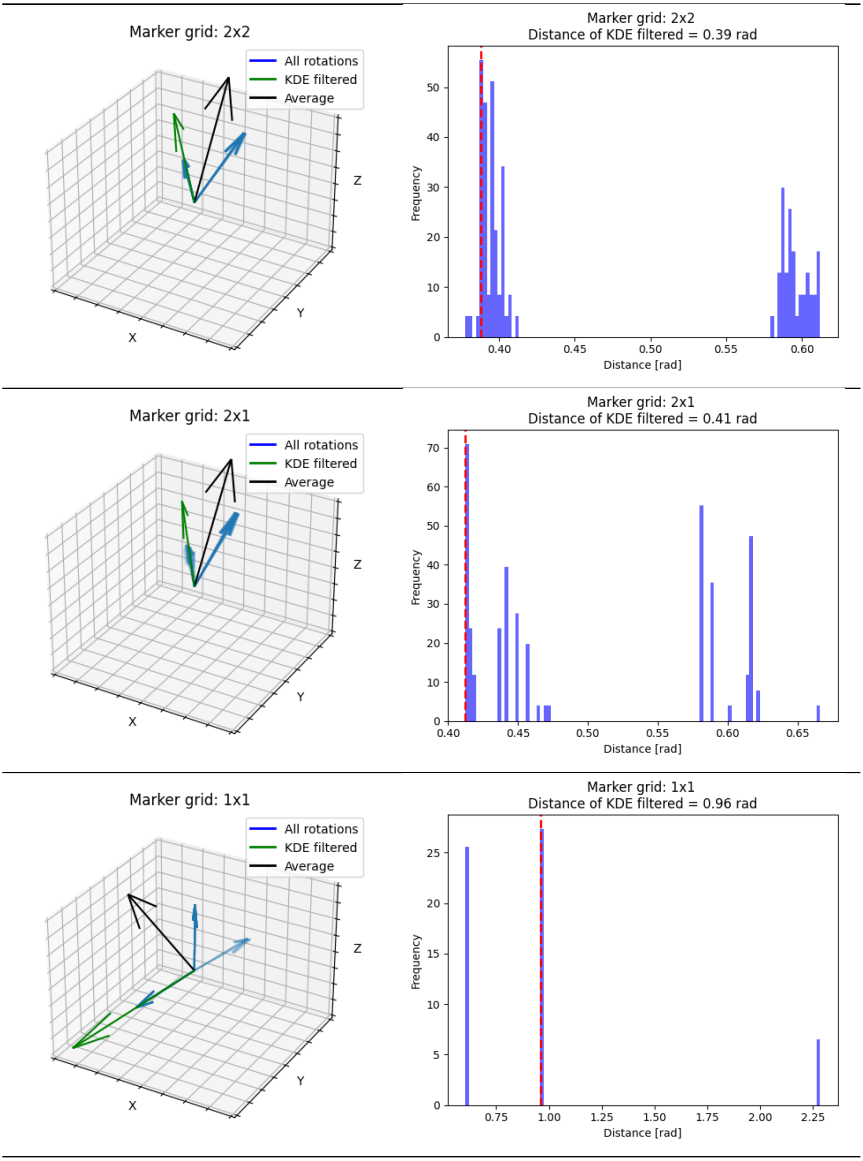
Figure 5.2 Measured and expected discrepancies. Red dashed lines indicate the span of the typical working area in the robot cage. Black circles are the mean error of the measurements.

Table 5.1 Rotation results for pose estimation for different marker grids on the same static target. Each dataset contained 100 measurements. The left column shows a 3D plot of the rotations, the average rotation, and the result from the KDE filtering. The average and KDE-filtered rotation are drawn with increased length for better visualization. The right column shows a histogram of the deviations from the average rotation, with the deviation of the KDE filtered vector showed as the red dashed line.



Continued on next page

Table 5.1 – continued from previous page



Hand-eye calibration

Calculation result - The camera link pose was calculated, from the same recorded data, with the direct method and the methods implemented in OpenCV. The result, seen in Figure 5.4, shows how the methods by

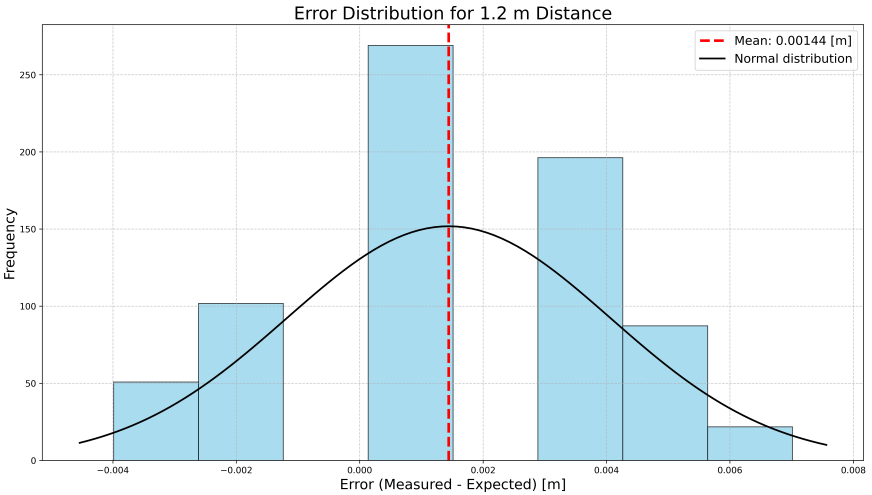


Figure 5.3 Distribution of errors at a distance of 1.2 m, with the mean value highlighted and featuring the distribution curve.

[Daniilidis, 1998], [Horaud and Dornaika, 1995] and [Tsai and Lenz, 1989] generated unusable poses. The methods by [Andreff et al., 1999] and [Park and Martin, 1994] generated more accurate but still unusable results as they were not accurate enough for sample handling. The direct method generated results with centimetre accuracy. After manual adjustment, a corrected version of the direct methods result was accurate enough for sample handling. In Figure 5.4, a corrected version is shown, with the translation adjusted by -3.3 mm in the X-direction and by -17.6 mm in the Y-direction.

Duration - The different execution times of a hand-eye calibration can be seen in Table 5.2. The total execution time for a 100 data-point calibration was 34 minutes and 13 seconds, with over 99.9% being associated with recording data. The result for different number of data-points can be seen in Figure 5.5.

Table 5.2 Breakdown of the execution times for the hand-eye calibration.

Step	Duration
Pick & Place of target	20.6 s
Record data	20.3 s / data-point
Calculation & Pose publishing	0.5 s
Total (100 data-points)	*34 min 13 s*

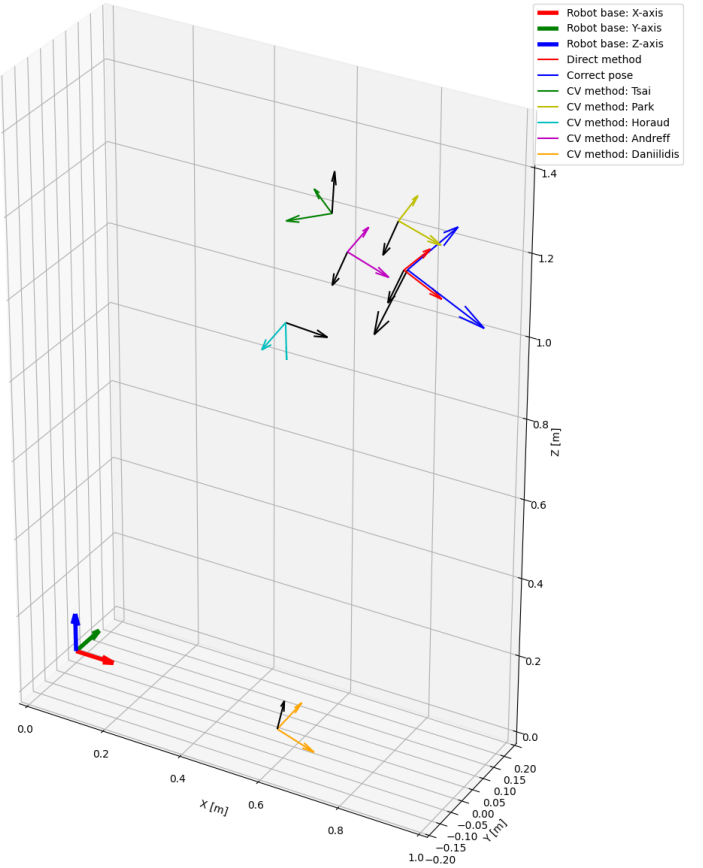


Figure 5.4 The pose of the camera link frame, relative the robot frame, for different hand-eye calibration methods. The direction the camera is facing is marked by the black axis. A manually corrected version, "Corrected", is also included (blue, with longer axes).

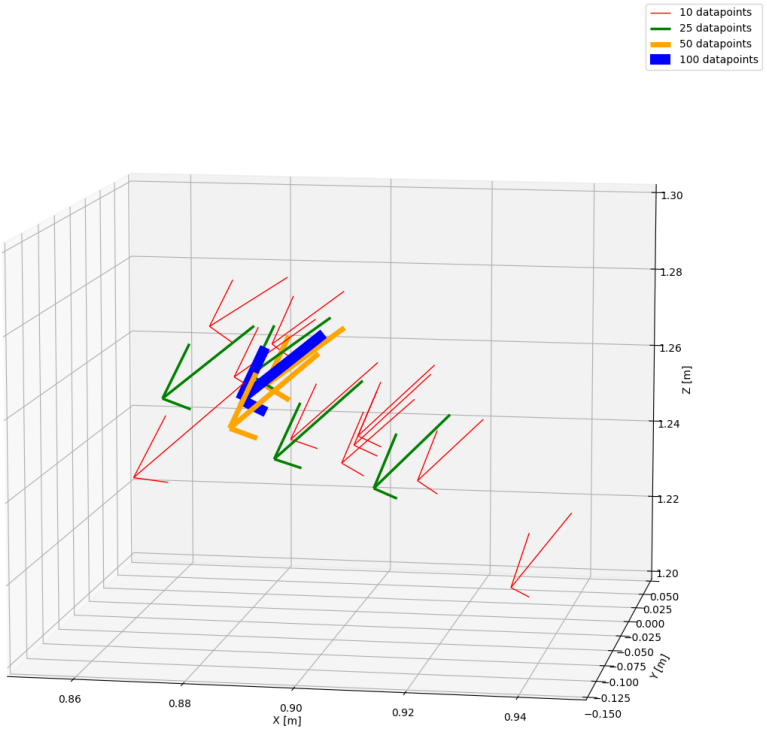
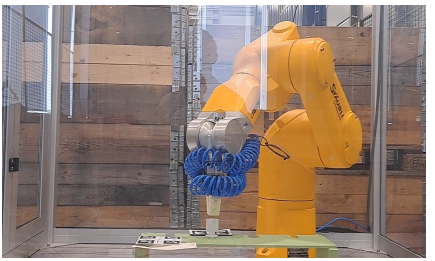


Figure 5.5 Hand-eye calibration result using the direct method for different number of data-points. The data is from one calibration and was sliced to get datasets of different sizes. The arrows in the plot are the x-axis of the camera optical frame.

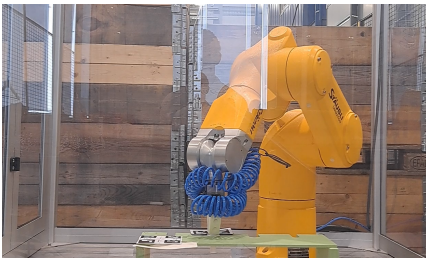
5.2 Complete System

The duration of a complete pick and place cycle, from the task is received until the robot is ready for a new task, between two different ports in the sample rack was measured to an average of 29.4 s. For a more complex task, the execution time increases, for example the task showed in Figure 5.6 took 40 s to execute, excluding the 5 s wait duration.

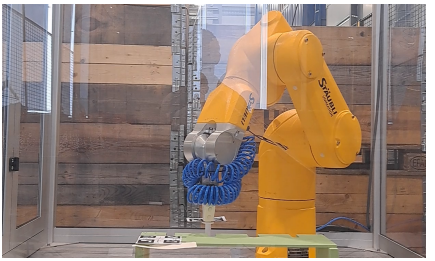
(a) `Home` - at the robot's waiting pose.



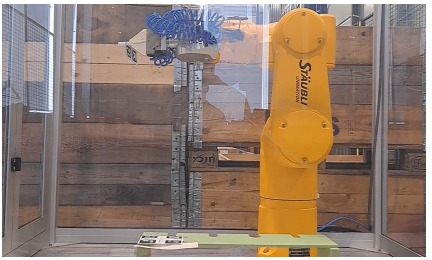
(b) `PickState` - at pre pose.



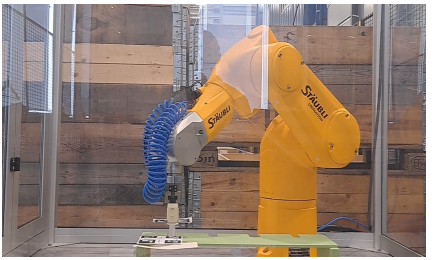
(c) `PickState` - at pick pose.



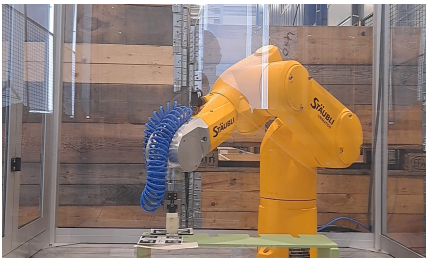
(d) `PickState` - at free pose.



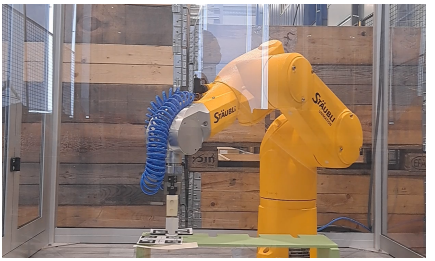
(e) `MoveToState` - at a user specified pose.



(f) `PlaceState` - at pre pose.



(g) `PlaceState` - at place pose.



(h) `PlaceState` - at free pose.

Figure 5.6 Photos from different stages of the task execution of a task compromised of the actions: `pick`, `move_to`, `wait` and `place`. Neither the pick nor the place pose was specified meaning the robot placed the sample in an available port that it identified.

6

Discussion

The goal of developing an autonomous system capable of automating the sample switching process was largely successful, with significant progress made within the thesis. However, further work is required before the system is ready for real operations. Many of the classes and methods developed throughout the thesis are general and can be used in a broader range of robotic applications. Key findings from this thesis include the development of a hand-eye calibration method that outperforms OpenCV's methods, and investigations into pose estimations and creation of filters to improve these estimations. From a system design perspective, a key finding is the approach taken in the design of this system, including the considerations necessary when developing an autonomous system and the methods for creating modular and general sub-systems.

6.1 Movement

By utilizing MoveIt and adding all objects in robot's environment to its planning scene, the robot was able to execute complex movement without collisions. This is crucial for applications in environments that could contain expensive scientific equipment. The integration with the robot was streamlined, thanks to the `staubli_val3_driver`, which was designed to work with MoveIt.

For this project, the high-level methods provided by MoveIt proved sufficient. However, MoveIt also allows for access to the different steps of the planning pipeline, enabling further control over the planning process and the ability to process intermediate results. This access allows for addressing future advanced optimization problems, such as jerk minimization which might become necessary for handling fragile materials.

All movement commands are abstracted by the `MGPI`, ensuring that additional features, such as the aforementioned optimization, can be implemented without requiring larger changes to the system. This shows how a modular design with clear responsibilities facilitates easy alterations and upgrades, which is essential for maintaining a flexible and adaptable system.

6.2 Perception

The perception part of the project was the sub-system that required the most development efforts, in particular to get sufficient pose estimation accuracy. There were several factors that contributed to measurements with insufficient accuracy, which are discussed more in-depth in this section.

Camera Calibration

An important breakthrough was the re-calibration of the camera. Initially, it was not believed that a calibration was necessary to get results sufficiently accurate for sample handling, however it turned out to be required due to a large absolute error at a distance. The issue was discovered when trying to pin-point the root cause of bad results from the hand-eye calibration. To characterise the depth error, an experiment was set up to measure the depth reading against known distances. This led to the collection of the datasets before calibration, seen in Figure 5.1. These showed that the depth errors were larger than the $\pm 2\%$ that was specified in the datasheet [*Intel® RealSense™ Product Family D400 Series* 2023]. This indicated that the issue could be due to either a poor base calibration or faulty sensors. To rule out the possibility of a calibration problem, a full re-calibration of the camera was performed. This significantly reduced the depth errors to an acceptable level, which allowed the development of the hand-eye calibration to proceed.

Pose Estimation

Filtering methods Concerning the pose estimation, gathering multiple measurements, and filtering proved critical for accurate estimations. As the distribution of the position and orientation measurements differed, different filtering techniques were utilized.

When examining the spread of the measurements in Figure 5.2, a considerable variation in the individual measurements can be observed. However, when the mean value of these measurements is computed, it is around ± 1 cm. Furthermore, when analysing a specific distance, in this case 1.2 m (chosen because it is the approximate distance from the cameras to the samples), it can be found that the distribution resembles a Gaussian distribution, whose mean accounts for an error of 1.44 mm. This is promising, as a Gaussian distribution means that a good estimation can be obtained by averaging the measurements. The overall implication here is that with simple filtering it is possible to obtain reasonable distance measurements from the camera, which is a crucial factor when estimating the pose of objects for the robot to pick.

As seen in Table 5.1, the orientation estimations had a multi-modal distribution, meaning averaging was not a valid way to produce better measurements. This can clearly be observed in the result as the average orientation is nowhere near any of the estimations. However, it can be seen that the more advanced filter based on Kernel Density Estimation, successfully managed to identify the most likely orientation,

corresponding to the highest peak in the histograms in Table 5.1. A key addition to the orientation filtering was adding a vertical constraint whenever possible. This solved the issue of the pitch and roll of the marker being difficult to estimate as they were determined by observing small perspective shifts.

Targets In Table 5.1 it can be seen how the modes become denser, meaning the estimations are more confident, for targets with more markers in their grid. This is partially because each marker provides an additional four points to the sample size for the PnP calculation, making the estimation more robust and accurate. Another reason is the increased overall size of the target, making perspective shifts from tilt and roll larger. Therefore, it was not surprising to discover that one marker was not enough for accurate orientation results, which could also be seen in the plot in Table 5.1. For objects where high accuracy was required, such as the hand-eye calibration target and the large sample rack, larger 2×2 target grids proved necessary. On the other hand, the pose of the small sample handles could be determined with enough accuracy with a smaller 2×1 grid, if they were guided by the vertical constraint.

Hand-Eye Calibration

Achieving an accurate hand-eye calibration was a prerequisite to get the perception system to work. Hand-eye calibration makes it possible to transform poses measured with reference to the camera's coordinate system to a coordinate system known by the robot. This also meant that if the hand-eye calibration was not good enough, then the pose estimation of samples would not be sufficiently accurate for pick and place. Therefore, an important finding from this thesis is illustrated in both Figure 5.4 and Figure 5.5. Our direct estimation method for determining the camera position outperformed all hand-eye calibration algorithms provided by OpenCV. However, it was not perfect and sometimes produced results that were one or two centimetres off, requiring minor manual adjustments before being accurate enough to handle samples.

Regarding the OpenCV methods, the methods by [Tsai and Lenz, 1989], [Horaud and Dornaika, 1995] and [Daniilidis, 1998] all produced results with significantly incorrect orientation and position, while the methods by [Andreff et al., 1999] and [Park and Martin, 1994] yielded results closer to the expected values. This discrepancy may be attributed to several factors, including the quality or quantity of the input data fed into the algorithms. Another potential cause is the optimization processes converging to local minima. Additionally, it is possible that the thesis' eye-to-hand configuration, compared to OpenCV's default eye-in-hand configuration introduces unexpected factors that these algorithms are not designed to handle. Although the input was adjusted to reflect the different configuration, and the fact that two of the methods produced results that were close to the actual solution, suggests that input errors are unlikely to be the primary cause.

When looking at Figure 5.5, it is evident that the hand-eye calibration results ex-

hibit a larger spread when using datasets with fewer data points. Conversely, as the number of data points increases, the result converges as expected. A limiting factor for the number of datapoints was the time required for data recording. With 20 seconds required per data-point, as shown in Table 5.2, recording 100 datapoints over a span of 34 minutes was deemed acceptable. However, for an operational installation, it may be desirable to gather more data for a more precise calibration. This could be achieved by accepting a longer data recording duration or by implementing a more efficient data gathering procedure.

6.3 System Design

The design of the autonomous system is a significant outcome of this thesis. The thesis provides insights into the considerations when developing such a system. The goal of developing a modular system, was achieved. By abstracting hardware specifics with general interfaces and wrapper classes, and leveraging the modular nature of ROS, a core program that is hardware agnostic was developed. This approach ensures that changing the robot arm or upgrading the camera only requires minor software adjustments or configuration changes. This flexibility is crucial for the future development and operational deployment of the system, as the system then would be unlikely to consist of the Stäubli TX60 robot arm or the RealSense D435if sensor. If the project had been developed in VAL3, it would have become dependent on Stäubli hardware, limiting its adaptability.

Another advantage is the utilization of widely known programming languages and tools. ROS allows programs to be written in Python or C++, languages that are well known at ESS. This increases the likelihood of the results being utilized and lowers the barrier of entry for future development.

The separation of the program into three distinct nodes: a movement node, a perception node, and a system control node, allowed for separate development and served to abstract the complex inner workings of each node. This design approach with abstractions, as discussed for the `MGPI` in Section 6.1, facilitates easy alterations and upgrades to individual parts of the system without necessitating alterations to the rest. This modularity enhances the system's maintainability and scalability, ensuring that it can be adapted to evolving requirements and technologies.

6.4 EPICS

The autonomous system was also designed to facilitate easier integration into EPICS. To leverage the power of ROS, all internal communication within the system was implemented through ROS, with top-level user input exposed for EPICS integration. User input is read in separate nodes and then passed to the system controller through services. This approach resulted in a significantly easier integration process, where the EPICS client and a ROS node were integrated. When the PV was

updated, the client listened for updates, extracted the raw values, created two ROS messages from these values, and sent them to the system control. To facilitate communication back to EPICS, a ROS message containing status information about the system can be implemented. This message would be translated into a process variable (PV) in the `ROS-client` node and subsequently update that PV. This approach allows users to track the system's progress from the Operator Interface (OPI) during operation.

Due to limited time, a large EPICS integration was not possible. Currently, predefined samples and tasks can be sent to the robot via an EPICS server-client using an OPI, demonstrating feasibility. In order for the current OPI to be a viable user interface for the system, further development is needed. The OPI should allow the addition of new samples and tasks, not just predefined items. Additionally, implementing a new ROS message type to send status updates back to the EPICS client-server for OPI display would enhance functionality and align with ESS deployment needs.

An observation that was made during the development of the system was that EPICS was not optimal for communicating with databases. This became a significant design choice when developing the system and was the reason the `SampleDB` was embedded in the ROS system, as opposed to exposing it to EPICS.

6.5 Sample Switching

The system is currently not robust enough to perform continuous unmonitored operation. This is due to inaccuracies in the perception, causing the robot to occasionally misalign with the sample handle during pick-up. When the system experiences a failure, it often autonomously, preventing the failures from being fatal to the system. For instance, if the system fails to place a sample in a rack port, it can attempt to place it in a different port. Similarly, if it encounters an issue partway through a pickup, it can reposition the sample before aborting. This self-recovery capability enhances the system's robustness and reliability.

Overall, the system demonstrates the feasibility of a more general approach for sample switching. The use of sample handles and racks allows for a wider range of sample shapes and sizes. Moreover, the system is flexible regarding the design of sample racks and handles, allowing for adjustments as needed when evaluating relevant samples for the application. Additionally, the system is flexible in the design of sample racks and handles, allowing for adjustments as needed when evaluating relevant samples for the application. If a new design accommodates even more shapes and sizes of samples, it can be developed and fitted with ArUco markers accordingly, without needing adjustments to the system.

6.6 Limitations

The development of this system faced several limitations, with time being one of the most significant constraints. Despite this, the goal of creating a flexible system for sample switching has been partially achieved.

While the use of ArUco markers on handles enabled the system to autonomously identify, pick, move, and place samples, it still requires user input to specify the tasks to be executed and describe the samples involved. Importantly, an STL file of the sample is required to ensure proper collision avoidance. However, it is expected that scientists will already have this file before arriving at ESS, and a coarse model is sufficient.

As mentioned, the camera performance is a limitation. The factory standard absolute depth error is $\pm 2\%$ at distances below 2 meters, which the filtering efforts have improved to an accuracy of around ± 1 cm. However, this error can still introduce uncertainty during operations. In environments such as at ESS or in industrial applications, higher precision may be required, necessitating the use of a different sensor.

The open-source drivers for the robot also proved a limitation as the drivers found where limited to ROS1. While still widely used, ROS1 will reach its End of Life in May 2025, making a transition to ROS2 desirable.

6.7 Implications

This thesis highlights several aspects, the most obvious being the difficulty of perception problems. Even when using high-contrast ArUco markers and readily available pose estimation methods, obtaining accurate measurements is challenging. While sensor performance is critical for achieving good results, filtering and other algorithms have been demonstrated to partially compensate for sensor limitations. Additionally, the developed custom hand-eye calibration method managed to outperform the OpenCV methods. This implies that developing specialized methods for hand-eye calibration, rather than relying solely on open-source methods, can be beneficial from a performance and control viewpoint.

6.8 Research Questions

Robotic Automation and System Integration

Looping back to the research question, it was deliberately chosen to develop the autonomous system that centralizes the core logic in a single computer, as illustrated in Figure 3.5. While the system is running on two devices-one computer and the CS8c controller-the computer handles the main system logic, and the communication with the robot controller. By focusing the logic on the computer, the system

simplifies integration with external systems like EPICS, as the single computer acts as the main interface point for such integrations.

The system was designed to enable user input from outside the ROS environment, facilitating straightforward integration with EPICS. This integration is achieved through an EPICS client that also functions as a ROS node, translating incoming EPICS PVs into a ROS request message that it sends to the system controller. Basic data types (strings, integers, and floats) are used for inputs, with specific ROS messages created automatically. Using basic data types for system inputs simplifies integration, eliminating the need for developing advanced drivers. This approach enhances the efficiency and adaptability of the system, ensuring communication between different components and user interfaces.

As mentioned, it would be desirable to have functionality for sending status updates back to EPICS. Implementing this would not be difficult, as the execution process is tracked, allowing for simple publishing of messages when changes occur. These messages could then be read by the EPICS node and forwarded to the operator. However, communication within the system should remain through ROS, as it is a more powerful tool for robotics than EPICS.

By using the developed autonomous system, it is possible to perform a more diverse set of experiments, which is desirable from a research standpoint. Beam time will be expensive for visiting scientists, and it will be important to maximize the output from the neutron beam. Considering, more functionality such as control based on experimental feedback should be implemented. This means that the robot receives feedback from the experiment while it is running, adjusting the sample to get the highest possible resolution from the neutrons.

Precision and Efficiency in Robotic Operations

Regarding the precision and efficiency, it is evident from the results shown in Figure 5.2 that there is a lack of precision in the sensor performance of the camera. Replacing it with a more industrial-grade model could significantly enhance this performance. The entire system is designed in a modular approach, which makes it relatively easy to change the camera. A sensor publishing onto the same topics would require no code alterations, while a completely different sensor would only require changes to the `MarkerDetector` class. This was a deliberate design choice to facilitate easy sensor upgrades.

As discussed, the sensor's precision is not sufficiently satisfactory, with the camera's depth measurement accuracy around ± 1 cm. When examining the Stäubli datasheet in Table 3.1, the robot's repeatability is shown to be ± 0.02 mm, which is significantly more precise than the camera. This indicates room for improvement, as the perception system's accuracy currently becomes the bottleneck of the system.

Unfortunately, there was no access to precision instruments that could precisely measure the repeatability of the system, this made it challenging to evaluate the exact precision of the robot's movements. The greatest bottleneck in this case is the

pose estimation, as an accuracy deviation of around 1 cm can occasionally cause the robot to fail its tasks. Additionally, the gripper's gap is not much wider than the sample handle when open, leaving little margin for error when gripping objects. This issue could be partially mitigated with a redesigned handle, but it would not address the underlying problem of measurement accuracy.

The cycle time of the system is approximately 30 seconds, from the home position back to the home position. The choice of this specific home state significantly impacts the cycle speed; however, only small adjustments is needed in case another home position is more beneficial. It is important to note that this cycle time does not include holding the sample in a hypothetical neutron beam. Instead, it illustrates the time required for the pick-and-place operation. This cycle time imposes some restrictions on how many cycles can be performed within a given time. Currently, there is no established duration for a neutron instrument investigation. However, depending on this duration, the movement may need to be optimized to shorten the cycle time.

Recognition and Positioning Techniques for Robotic Systems

Regarding this recognition and position techniques, in both research and industrial applications, there is often a level of environmental predictability. This predictability necessitates trade-offs in the development of autonomous systems, similar to any design and development process. Developing a fully autonomous system may not always be practical, primarily due to the challenges associated with perception. Creating a general-purpose vision model that can detect objects and estimate their poses with the required precision is difficult. Moreover, designing such a model to be so general that it can handle any object—especially given the wide variety of samples in the ESS environment—is even more challenging due to the inherent complexities of machine learning.

This led to the design of sample racks and handles, with ArUco markers naturally evolving from this concept. By utilizing these markers, it is possible to give the samples known key points, which simplifies the object detection and pose estimation by searching in the images after the known patterns. Scientists need only to attach samples to the sample handles, after which the system takes over, managing the remaining part of the process autonomously. This approach balances the need for precision and efficiency while simplifying the integration of autonomous systems into existing workflows.

6.9 ESS Goals

Transferring Robotics Knowledge to ICS One of the goals for ESS was the transfer of robotics knowledge to the ICS division. To achieve this, teaching sessions were organized for the ESS supervisors and presentations were held for the division.

These sessions provided the supervisors with a deep understanding of the system, ensuring they are well-equipped to continue the development work independently.

In addition to teaching sessions, the work has been thoroughly documented. This documentation includes detailed instructions for setting-up and running the system, comprehensive system diagrams and logging of activities along with the reasoning behind key decisions. This report also serves as a critical component of the knowledge transfer process. The codebase itself has been crafted with a strong emphasis on clarity and ease of understanding. This was achieved through documentation, consistent commenting, and a well-defined system design.

By focusing on these aspects, a solid foundation for future development has been laid. Ensuring that ESS and the ICS division can continue exploring the use of robotics in research applications.

Feasibility of EPICS Integration The feasibility of integrating EPICS into the system has been successfully demonstrated. The current setup allows tasks and sample information to be sent to the system via EPICS. Adding functionality for passing information from the system controller back to EPICS will require little effort, and there are no technical obstacles for implementing it. The current EPICS drivers and code can serve as guidelines for the addition of further functionality.

6.10 Future Work

Vision Based Closed Loop Control

For increased accuracy and reliability of the robot's movement, incorporating a vision based closed-loop control system could be beneficial. Currently, the perception system only provides the goal for the movement without monitoring the execution process. By incorporating a feedback loop using position-based visual servoing (PBVS) and continuous pose estimations of the sample, the robot can adjust its movement based on real-time data from the perception system, allowing it to pick and place samples even if the calibration or initial pose estimation is slightly inaccurate. This approach could also enable the robot to achieve movement precision beyond its specified capabilities.

Instrument Integration

For the system to be effectively used within an instrument at ESS or a similar facility, additional functionalities is needed. More actions, in addition to the implemented `move_to`, should be added for controlled movement within the instrument, such as moving it along a predetermined trajectory.

Furthermore, incorporating control based on real-time feedback from the instrument's measurements is a very interesting use of this system. This would enable the robot to adjust the sample position dynamically, based on the continuous results of the experiments, ensuring that data is gathered from the most relevant point at all

times. This adaptive approach could significantly increase the efficiency of scientific experiments.

Further EPICS Integration

There is also work to be done on the EPICS integration. The system should send updates back to EPICS, so that users can supervise the robot while it is in operation. Ideally users should be able to see camera feeds, visualize and confirm movement like in RViz, and manually control the robot. Users should also be able to input new sample and task types straight into the OPI, and not depend on updating the JSON file.

7

Conclusions

This thesis presents the design and development of an autonomous system for flexible sample switching at the European Spallation Source ERIC. The developed system has demonstrated that a computer vision based system can be used for sample switching.

Additionally, it shows how an external computer with ROS can control a TX60 robotic arm, interface with sensors such as the Intel RealSense D435if depth camera and receive input from an external control system like EPICS. This integration highlights the modular and adaptable nature of the system, facilitating seamless operation and coordination among components.

The system, after camera calibration and with the aid of pose filtering, proved capable of picking up, manoeuvring and placing samples. However, the use of more advanced sensors could improve the operational efficiency and robustness of the system. Enhancing the accuracy of the computer vision system will increase the number of completed cycles. Additionally, a more precise system will enable the incorporation of functionalities such as fine adjustments within the neutron beam.

Furthermore, it has been concluded that, for this experimental setup, the custom direct hand-eye calibration method can outperform the methods implemented in OpenCV.

Future work should focus on three key areas to further enhance the system's capabilities. First, developing a closed-loop control system based on real-time feedback from the perception system could enable more precise movements and adjustments during sample handling. Secondly, integrating the system with the neutron instruments to receive feedback based on experimental results could significantly increase the operational efficiency and scientific output. Lastly, further developing the integration with EPICS should be a priority, to enable full control of the robotic system through ESS's control system.

This thesis contributes to the robotics field by providing a study on how to design and develop a versatile and adaptable autonomous system, highlighting challenges and solution in various areas in robotics. Additionally, the experimental results show that the developed system demonstrates promise for operational sample switching by completing full cycles from detection to actuation.

Bibliography

About EPICS (n.d.). https://epics-controls.org/about-epics/. Accessed: 2024-05-25.

About ESS (n.d.). https://europeanspallationsource.se/about. Accessed: 2024-05-26.

Andreff, N., R. Horaud, and B. Espiau (1999). "On-line hand-eye calibration". In: *Proceedings of the 2nd International Conference on 3-D Digital Imaging and Modeling (3DIM'99)*. IEEE Computer Society, Washington, DC, USA, pp. 430–436.

ArucoDetector Class Reference (n.d.). https://docs.opencv.org/4.x/d2/d1a/classcv_1_1aruco_1_1ArucoDetector.html. Accessed: 2024-05-24.

Bohren, J. (2018). *Smach - ros wiki*. http://wiki.ros.org/smach. Accessed: 2024-02-10.

Camera Calibration and 3D Reconstruction (n.d.). https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html. Accessed: 2024-05-24.

CameraInfo.msg (2022). https://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/CameraInfo.html. Accessed: 2024-05-27.

Chen, Y.-C. (2017). "A tutorial on kernel density estimation and recent advances". *Biostatistics & Epidemiology* **1**:1, pp. 161–187. DOI: 10.1080/24709360.2017.1396742.

Conley, K., D. Thomas, and J. Perron (2019). *Roslaunch- ros wiki*. http://wiki.ros.org/roslaunch. Accessed: 2024-05-24.

Damaševičius, R., R. Maskeliunas, G. Narvydas, R. Narbutaite, D. Połap, and M. Woźniak (2020). "Intelligent automation of dental material analysis using robotic arm with jerk optimized trajectory". *Journal of Ambient Intelligence and Humanized Computing* **11**. DOI: 10.1007/s12652-020-02605-8.

Daniilidis, K. (1998). "Hand-eye calibration using dual quaternions". *International Journal of Robotics Research* **18**, pp. 286–298.

Davidsaver, M. (2023). *Pvaccess for python (p4p)*. https://mdavidsaver.github.io/p4p/. Accessed: 2024-05-26.

Detection of ArUco Markers (n.d.). https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html. Accessed: 2024-05-24.

Digumarti, T. (2023). *Movegrouppythoninterface class to control the ur5e arm using ros and moveit!* mailto:tejaswi.digumarti@sydney.edu.au. Unpublished code from the Unit MTRX5700, The University of Sydney.

Foote, T., E. Marder-Eppstein, and W. Meeussen (2019). *Tf2 - ros wiki*. http://wiki.ros.org/tf2. Accessed: 2024-05-24.

GitHub (n.d.). *Copilot*. https://github.com/features/copilot. Accessed: 2024-05-22.

Grunnet-Jepsen, A., J. Sweetser, T. Khuong, S. Dorodnicov, D. Tong, O. Mulla, H. Eliyahu, and E. Raikhel (n.d.). *Intel® RealSense™ Self-Calibration for D400 Series Depth Cameras*. Rev 2.7. Available at https://dev.intelrealsense.com/docs/self-calibration-for-depth-cameras.

Hemayed, E. (2003). "A survey of camera self-calibration". In: *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance, 2003.* Pp. 351–357. DOI: 10.1109/AVSS.2003.1217942.

Hershberger, D., D. Gossow, J. Faust, and W. Woodall (2018). *Rviz - ros wiki*. http://wiki.ros.org/rviz. Accessed: 2024-05-24.

Heyer, C. (2010). "Human-robot interaction and future industrial robotics applications". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4749–4754. DOI: 10.1109/IROS.2010.5651294.

Horaud, R. and F. Dornaika (1995). "Hand-eye calibration". *International Journal of Robotics Research* **14**:3, pp. 195–210. DOI: 10.1177/027836499501400301.

How the Accelerator works (n.d.). https://europeanspallationsource.se/accelerator/how-it-works. Accessed: 2024-05-26.

Intel® RealSense™ Product Family D400 Series (2023). Rev. 017. Data sheet no. 337029-017. Available at https://www.intelrealsense.com/download/21345/?tmstv=1697035582.

Intel® RealSense™ SDK 2.0 (2024). https://github.com/IntelRealSense/librealsense. Accessed: 2024-02-21.

Introduction - ROS Wiki (2018). https://wiki.ros.org/ROS/Introduction. Accessed: 2024-05-26.

Jocher, G., A. Chaurasia, and J. Qiu (2023). *Ultralytics YOLO*. Version 8.0.0. URL: https://github.com/ultralytics/ultralytics.

Kotthauser, T. and G. F. Mauer (2009). "Vision-based autonomous robot control for pick and place operations". In: *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 1851–1855. DOI: 10.1109/AIM.2009.5229792.

Lynch, K. and F. Park (2017). *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press. ISBN: 9781316609842. URL: https://books.google.com/books?id=86G0nQAACAAJ.

Marder-Eppstein, E., V. Pradeep, and M. Arguedas (2024). *Actionlib - ros wiki*. http://wiki.ros.org/actionlib. Accessed: 2024-05-24.

Martins, M. (2020). *Staubli_val3_driver - ros wiki*. https://wiki.ros.org/staubli_val3_driver. Accessed: 2024-01-16.

Messages - ROS Wiki (2016). http://wiki.ros.org/Messages. Accessed: 2024-05-24.

MoveIt (n.d.). https://moveit.ros.org. Accessed: 2024-01-23.

MoveIt Setup Assistant (n.d.). https://docs.ros.org/en/kinetic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html. Accessed: 2024-01-23.

Nodes - ROS Wiki (2018). http://wiki.ros.org/Nodes. Accessed: 2024-05-24.

OpenAI (n.d.). *Chatgpt: language model*. https://www.openai.com/chatgpt. Accessed: 2024-05-22.

Park, F. C. and B. J. Martin (1994). "Robot sensor calibration: solving AX=BX on the Euclidean group". *IEEE Transactions on Robotics and Automation* **10**:5, pp. 717–721.

Peggs, S. et al. (2012). *Conceptual Design Report*. Tech. rep. European Spallation Source, Lund, Sweden, p. 71. URL: https://europeanspallationsource.se/sites/default/files/downloads/2017/09/CDR_final_120206.pdf.

Perspective-n-Point (PnP) pose computation (n.d.). https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html. Accessed: 2024-05-24.

Phoebus 1.0 Documentation (n.d.). https://control-system-studio.readthedocs.io/en/latest/intro.html. Accessed: 2024-05-21.

Pooley, A. and M. Lautman (2013). *Move group python interface tutorial*. https://github.com/moveit/moveit_tutorials/blob/kinetic-devel/doc/move_group_python_interface/scripts/move_group_python_interface_tutorial.py. Accessed: 2024-02-02.

Popovic, M. B. and M. P. Bowers (2019). "2 - kinematics and dynamics". In: Popovic, M. B. (Ed.). *Biomechatronics*. Academic Press, pp. 11–43. ISBN: 978-0-12-812939-5. DOI: https://doi.org/10.1016/B978-0-12-812939-5.00002-1. URL: https://www.sciencedirect.com/science/article/pii/B9780128129395000021.

Premebida, C., R. Ambrus, and Z.-C. Marton (2018). "Intelligent robotic perception systems". *Applications of mobile robots*, pp. 111–127.

Randau, C., H. Brokmeier, W. Gan, M. Hofmann, M. Voeller, W. Tekouo, N. Alhamdany, G. Seidl, and A. Schreyer (2015). "Improved sample manipulation at the stress-spec neutron diffractometer using an industrial 6-axis robot for texture and strain analyses". *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **794**, pp. 67–75. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2015.05.014.

ros-realsense (2024). https://github.com/IntelRealSense/realsense-ros/tree/ros1-legacy. Accessed: 2024-02-21.

Schmid, A. J., L. Wiehemeier, S. Jaksch, H. Schneider, A. Hiess, T. Bögershausen, T. Widmann, J. Reitenbach, L. P. Kreuzer, M. Kühnhammer, O. Löhmann, G. Brandl, H. Frielinghaus, P. Müller-Buschbaum, R. von Klitzing, and T. Hellweg (2021). "Flexible sample environments for the investigation of soft matter at the european spallation source: part i—the in situ sans/dls setup". *Applied Sciences* **11**:9. ISSN: 2076-3417. DOI: 10.3390/app11094089. URL: https://www.mdpi.com/2076-3417/11/9/4089.

Services - ROS Wiki (2019). http://wiki.ros.org/Services. Accessed: 2024-05-24.

Stäubli (n.d.). https://www.staubli.com. Accessed: 2024-04-14.

Stäubli (2007). *TX60 series industrial robots*. Data sheet no. D181.541.04. Pdf shipped with robot.

Tsai, R. Y. and R. K. Lenz (1989). "A new technique for fully autonomous and efficient 3D robotics hand/eye calibration". *IEEE Transactions on Robotics and Automation* **5**:3, pp. 345–358.

VAL3 language (n.d.). https://www.staubli.com/hk/en/robotics/products/robot-software/staeubli-robotics-controls/val-3-language.html. Accessed: 2024-05-25.

vd. Hoorn, G. (2018). *Staubli_experimental - ros wiki*. http://wiki.ros.org/staubli_experimental. Accessed: 2024-01-23.

Wang, C., D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese (2019). "Densefusion: 6d object pose estimation by iterative dense fusion".

Wilson, P. and H. A. Mantooth (2013). "Chapter 6 - block diagram modeling and system analysis". In: Wilson, P. et al. (Eds.). *Model-Based Engineering for Complex Electronic Systems*. Newnes, Oxford, pp. 169–196. ISBN: 978-0-12-385085-0. DOI: https://doi.org/10.1016/B978-0-12-385085-0.00006-3. URL: https://www.sciencedirect.com/science/article/pii/B9780123850850000063.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
June 2024

Document Number
TFRT-6247

Author(s)
Anthon Andersson
Anton Håkansson

Supervisor
Tomasz Bryś, European Spallation Source ERIC, Sweden
Karin Rathsman, European Spallation Source ERIC, Sweden
Björn Olofsson, Dept. of Automatic Control, Lund University, Sweden
Yiannis Karayiannidis, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Flexible Computer Vision based Sample Switching System using a Robotic Arm

Abstract

This thesis presents the development of a computer vision based robotic system for flexible sample switching at the European Spallation Source (ESS). At ESS, samples will be placed in the neutron beam by the instruments. These environments will be exposed to radiation, making direct access to the samples difficult for personnel. To maximize scientific output, it is also desirable to minimize the downtime of the neutron beam. These factors make an automated solution for sample switching highly desirable. Traditional robotic systems often lack the adaptability needed to perform in settings that are subject to change. Consequently, it is hard to develop robotic systems that are general enough to be independent of a specific robotic arm and the environment, using the traditional approach. This project addresses these limitations by creating a robotic system capable of operating autonomously in a dynamic environment.

The system integrates several components: a Stäubli TX60 industrial robot, an Intel RealSense D435if depth camera, and custom 3D-printed sample handles with ArUco markers for identification and pose estimation. ROS is used for control and perception, and EPICS drivers is developed for integration with ESS's control system.

The results demonstrate the feasibility of integrating autonomous robotic systems into complex research environments, showing promise in improving operational functionality in sample handling at ESS. Future work will focus on enhancing system robustness, upgrading hardware, expanding functionality, and further integration with ESS's control systems.

Overall, this thesis contributes to the greater field of robotics by providing a case study of a versatile and adaptable robotic system, highlighting challenges and solutions in developing autonomous systems for scientific research.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-67

Recipient's notes

Security classification

http://www.control.lth.se/publications/