MASTER'S THESIS 2023

Skill-Based Autonomous Door Opening

Josefin Gustafsson, Pontus Rosqvist

ISSN 1650-2884
LU-CS-EX: 2023-32

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE

Datavetenskap

LU-CS-EX: 2023-32

Skill-Based Autonomous Door Opening

Färdighetsbaserad autonom dörröppning

Josefin Gustafsson, Pontus Rosqvist

Skill-Based Autonomous Door Opening

Josefin Gustafsson
josefingustafsson98@hotmail.com

Pontus Rosqvist
pontus.rosqvist@gmail.com

June 30, 2023

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Volker Krüger, volker.krueger@cs.lth.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

A generalizable skill which is able to open a door operated by a button is specified and implemented on an autonomous robot. This is achieved using ROS and SkiROS on a mobile robot called Heron. A skill which lets Heron pass the door while ensuring that the door stays open is also specified and implemented. A building is described with a graph structure which enables Heron to plan a path which will let it navigate to a goal location from any start location in the building.

Physical experiments have been performed to verify that the expected behaviour matches the actual behaviour of the skills. These experiments were successful, recorded and included in the appendix.

Keywords: Robot, Robot Skills, SkiROS, ROS, Door Opening

Acknowledgements

This project was done for the division of Robotics and Semantic Systems at the Faculty of Engineering, Lund University. We would like to thank all of the employees there for their help.

We would like to thank Volker Krüger, our supervisor, for guidance and help on the project.

We would like to thank Simon Kristoffersson Lind for all of the support, discussions and help throughout the project. Finally we would also like to thank Ayesha Jena for her invaluable company and support.

Josefin Gustafsson and Pontus Rosqvist

Contents

Chapter 1

Introduction

1.1 Background

Autonomous systems can be defined as "systems that can change their behavior in response to unanticipated events during operation" [28]. Compare this to the majority of robots on production lines which may struggle to manage unforeseen circumstances or even be unaware that such situations have occurred. Autonomous robots possess a greater level of adaptability and can respond effectively in dynamic environments.

Industrial robots are able to consistently perform the same task but do not sense their general surroundings, for this reason they are confined to operating in closed environments free of humans. Autonomous robots can be used where traditional non-perceiving robots could never be utilized. They can be used in environments where inconsistencies are unavoidable. For example the robot can not expect an object to stay exactly where it was previously detected.

Any task an autonomous robot needs to perform can be broken up into three parts, sensing, planning and acting. A simpler robot often uses sensing and acting but leaves the planning part to a human, this plan is often pre-specified and can not be changed at runtime. A fully autonomous robot is able to sense it's surroundings and depending on the results it can plan how it should achieve a certain goal and then act in the environment to reach that goal without any human intervention. [24]

Autonomous robots can be utilized in many situations and help humans with tasks that they themselves can not perform. The need for autonomous robots can be due to a number of different reasons. These robots should not require new environments for us to be able to

utilize them, they should meet humans where they are [4, p.52]. The majority of human activities occur within buildings. It follows that for a robot to be able to complete tasks there they must possess the capability to navigate through these structures.

Traversing a building requires a robot to posses many different skills e.g. using an elevator or walking up a flight of stairs. The skills the robots use to navigate with exist purely in service of the bigger task, they do not directly help it complete the task but could be a necessity for reaching the goal. This because many actions require movement, sometimes between rooms and or floors.

1.2 Problem Description

From the previous discussion it is clear that a collaborative autonomous robot needs to posses several different skills which enable it to navigate. Of these a door opening skill is one of the most useful to navigate in a building. This project will focus on implementing a general door opening skill for a specific type of door and a specific robot.

A door opening skill can be very different depending on what the door looks like and how it is operated. For this project the chosen door can be opened by pressing a button. The specific task the robot should complete will be to navigate to the button, press the button, verify that the door is open and go through it.

The robot we will use is called Heron and consists of a MiR platform, a realsense D435 depth camera and a UR5e arm with a WSG 50-110 gripper (an end-effector). Heron can be seen in Fig. 1.1.

1.3 Research Questions

For us to program a robot such that it has a door opening skill we first need to define what a door opening skill is or even what it should look like. What factors are important for the robot when opening a door? What conditions should hold before the skill is executed and what conditions should be true after the skill has been executed?

In this specific scenario we need to implement a skill that Heron can execute that opens the specific door we have in mind. The skill should however be generalized so that it is able to open other doors.

From this the following research questions can be formulated:

- How should a door opening skill be specified?
- What conditions does a door opening skill need to be satisified?
- How can a door opening skill be implemented for Heron?

Figure 1.1: The robot, Heron, which will be used in this project.

- Can we utilize fiducial markers to reliably detect the button?
- How consistent and precise is the manipulation of the arm?
- How consistent and precise is the robot navigation?
- How reliable is the force sensing?
- How reliable is the button pushing as a whole?
- Can Heron itself plan how a door opening skill should be utilized to help it navigate in the environment?

1.4 Related Work

Robust and Adaptive Door Operation with a Mobile Robot

Using robots to support humans means that the robot should be reliable and efficient. Arduengo et al. have, in their research paper used YOLO on RGB-D images together with efficient point-cloud processing to detect doors and handles. [3]

They additionally propose a Bayesian framework that enables the robot to infer how the door is operated, if it is a cabinet you pull open or a door which swings open. Arduengo et al. have then combined these techniques to perform real-world experiments using a robot.

Force-Vision Sensor Fusion Improves Learning-Based Approach for Self-Closing Door Pulling

Opening a self-closing door is considerably harder than opening a door which does not close on its own. As the robot passes through the door it needs to hold the door open and make sure that it has passed the door before the door closes.

Sun et al. have used deep reinforcement learning and vision sensor fusion to simulate what pulling and passing a self-closing door could look like. [25]

OpenD: A Benchmark for Language-Driven Door and Drawer Opening

Assisting humans does not only include traversing a building, it can entail fetching things. This could make it necessary to open things other than doors.

OpenD is a simulation environment which includes many different door and drawer opening environments with different types of grippers. Zhao et al. desribe a multi-step plan for how a cabinet opening skill could be formulated and train several models which are able to open different kinds of doors and cabinets. [29]

Planning for Autonomous Door Opening with a Mobile Manipulator

Chitta et al. recognize that determining a skill sequence for opening a general door at runtime involves a search space which is so large that the problem becomes intractable. By discretizing the problem and introducing a graph structure they are able to decrease the search space and efficently determine the sequence of skills which needs to be executed to achieve their goal. [6]

Door Manipulation with the Spot Arm

Boston Dynamics have produced the famous Spot robot which is able to perform many different tasks. One of these tasks is a door opening task for a very specific type of door with a door handle. [9]

Chapter 2

Theory

2.1 Perception

2.1.1 Homogeneous Coordinates

Homogeneous coordinates are a useful representation of vectors from projective geometry that are defined up to scale. This is especially useful in computer vision where projections of points to a camera plane can be simplified greatly with homogeneous coordinates. The space $\mathbb{P}^n$ is defined to be the n-dimensional projective space. This can be identified with the $n+1$ dimensional real space $\mathbb{R}^{n+1}$ with the origin removed [12, p.29].

A very important note about projective space is that it is defined up to scale, we say that points are equal if they only differ in scale, i.e.

$$\mathbf{x} \sim \mathbf{y} \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^{n+1} \qquad \Longleftrightarrow \qquad \mathbf{x} = \lambda \mathbf{y} \text{ for some } \lambda \in \mathbb{R}, \lambda \neq 0 \ .$$

This space can be defined for any amount of dimensions but in computer vision one uses $\mathbb{P}^2$ to represent points on the camera plane and $\mathbb{P}^3$ to represent points in the world. We can map vectors in $\mathbb{R}^n$ to and from $\mathbb{P}^n$ with the following functions

$$f : \mathbb{R}^n \to \mathbb{P}^n,$$
$$g : \mathbb{P}^n \to \mathbb{R}^n.$$

We define these functions in the following way

$$f : \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \mapsto \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \\ 1 \end{bmatrix},$$

$$g : \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \\ v_{n+1} \end{bmatrix} \mapsto \frac{1}{v_{n+1}} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}.$$

If we use homogeneous coordinates the Euclidean coordinates of points in the camera frame are related to the homogeneous coordinates of their projections to the camera plane with the following equation

$$\begin{bmatrix} x_{\text{pix}} \\ y_{\text{pix}} \\ 1 \end{bmatrix} \sim \begin{bmatrix} X_{\text{cam}} \\ Y_{\text{cam}} \\ Z_{\text{cam}} \end{bmatrix} \quad .$$

To recover x_{pix} and y_{pix} we can apply g, i.e. compute the following [12, p.27]

$$x_{\text{pix}} = \frac{X_{\text{cam}}}{Z_{\text{cam}}} \quad ,$$
$$y_{\text{pix}} = \frac{Y_{\text{cam}}}{Z_{\text{cam}}} \quad .$$

In projective geometry we refer to vectors that are normalized, such that the last coordinate is 1, as inhomogeneous vectors, while vectors that are not normalized in this way are homogeneous vectors [26, p.36].

Homogeneous coordinates are a powerful way to describe coordinates and are widely used in computer vision and photogrammetry. This is due to the fact that imaging points requires projecting them to a plane which can be done efficiently with this framework.

2.1.2 Coordinate Transformations

Given a three-dimensional vector, $\mathbf{x}$, and two bases $\{\mathbf{e}_i\}_{i=1}^3$ and $\{\mathbf{f}_i\}_{i=1}^3$, we can represent x by coordinates in these bases in the following way

$$\begin{aligned} \mathbf{x} &= x_1\mathbf{e}_1 + x_2\mathbf{e}_2 + x_3\mathbf{e}_3 \\ &= x_1'\mathbf{f}_1 + x_2'\mathbf{f}_2 + x_3'\mathbf{f}_3 \quad . \end{aligned}$$

The coordinates are not necessarily equal but describe the same point. Clearly there is no reason to choose one basis over the other as a canonical basis, but both of these bases have the same origin. To see this consider how both of these bases would represent 0.

By the same token there is no reason to choose any point as the origin over any other. This decoupling of points from their coordinate system leads to the theory of homogeneous coordinate transformations.

Since a point in three-dimensional space still exists and has a known position relative to other points even if we do not give it explicit coordinates, one usually introduces coordinate frames which act as a fixed frame of reference. With these we can record how points relate to each other in the physical space. When noting the coordinates of a point this is always in relation to some frame which we can see in Fig. 2.1.

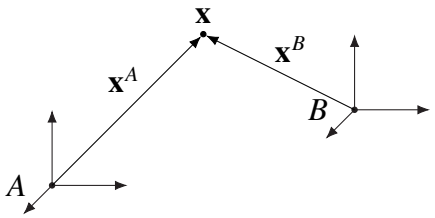

Figure 2.1: One point in physical space described in coordinate frame A and coordinate frame B. The vector $\mathbf{x}^A \in \mathbb{R}^3$ is the coordinates of $\mathbf{x}$ in frame A and $\mathbf{x}^B \in \mathbb{R}^3$ is the coordinates of $\mathbf{x}$ in frame B.

The point $\mathbf{x}$ in Fig. 2.1 has a representation in frame A given by a set of coordinates denoted by the vector $\mathbf{x}^A$ as well as a representation in frame B denoted by $\mathbf{x}^B$ [8, p.5]. Note that the coordinates $\mathbf{x}^A$ and $\mathbf{x}^B$ are not literally equal yet still describe the same point with respect to different frames.

We can also record the relation between different frames by recording where the origin of frame B gets placed in the coordinates of A. We call this vector $\mathbf{t}^A_{A\to B}$ where $A \to B$ denotes the translation that takes frame A to frame B and the superscript denotes which frame the translation has been recorded in. The orientation of the axes in coordinate frame B can also be recorded in frame A in a matrix $\mathbf{R}^A_{A\to B}$ which is an orthogonal matrix since its columns form an orthonormal basis. [15, p.35]

Given the translation, $\mathbf{t}^A_{A\to B}$, and rotation, $\mathbf{R}^A_{A\to B}$ that relate frame A to frame B we can convert the coordinates of a point from the coordinates in frame B to coordinates in frame A with the following formula

$$\mathbf{x}^A = \mathbf{R}^A_{A\to B}\mathbf{x}^B + \mathbf{t}^A_{A\to B} \ ,$$

note that this is an affine transformation i.e. not linear. This coordinate transformation can be reframed to a matrix product in homogeneous coordinates in the following way [15, p.36]. This rewriting turns the change of frame into a linear transformation in $\mathbb{P}^n$.

$$\begin{bmatrix} \mathbf{x}^A \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}^A_{A\to B} & \mathbf{t}^A_{A\to B} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}^B \\ 1 \end{bmatrix} \ .$$

The coordinate transformation from frame B to A can thus be described by the translation and rotation that moves the origin of frame A to frame B and aligns the axes of frame A with

the axes of frame $\boldsymbol{B}$. This with a single matrix [8, p.7]

$$\mathbf{T}_{A\to B}^{A} = \begin{bmatrix} \mathbf{R}_{A\to B}^{A} & \mathbf{t}_{A\to B}^{A} \\ 0 & 1 \end{bmatrix} \ . \tag{2.1}$$

When using homogeneous coordinate transformations there are some important rewrite rules that can be used to combine and modify coordinate transformations. Two of these rewrite rules are the following [15, p.37]

$$\left(\mathbf{T}_{A\to B}^{A}\right)^{-1} = \mathbf{T}_{B\to A}^{B} \ ,$$
$$\mathbf{T}_{C\to A}^{C} = \mathbf{T}_{C\to B}^{C}\mathbf{T}_{B\to A}^{B} \ .$$

In Fig. 2.2 we can see that computing the coordinates of $\mathbf{x}$ in frame $\boldsymbol{A}$ given the coordinates in frame $\boldsymbol{B}$ is similar to traversing the arrows from point $\mathbf{x}$ to frame $\boldsymbol{A}$. From this path we can reconstruct the formula $\mathbf{x}^{A} = \mathbf{T}_{A\to B}^{A}\mathbf{x}^{B}$. This way of viewing coordinate frames as nodes and coordinate transformations as connections between nodes is very powerful for computing coordinate transformations.

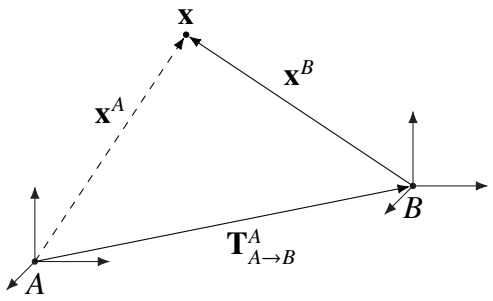

Figure 2.2: Computing the coordinates of $\mathbf{x}$ in frame $\boldsymbol{A}$ can be done by traversing the graph from $\mathbf{x}$ to $\boldsymbol{A}$ and reconstructing $\mathbf{x}^{A}$ from the determined path.

Another problem that can be effectively solved with this viewpoint is the problem of determining the transformation between two base frames if a single frame has been determined in both frames. The problem setup can be seen in Fig. 2.3.

Coordinate transformations can be determined by finding paths between frames in the graph of coordinate transformations. From this we can conclude that starting from frame $\boldsymbol{B}$ and going in a circle is identical to converting from frame $\boldsymbol{B}$ to itself, the identity transformation. This gives the following formula

$$\mathbf{T}_{B\to C}^{B}\left(\mathbf{T}_{A\to C}^{A}\right)^{-1}\mathbf{T}_{A\to B}^{A} = \mathbf{T}_{B\to B}^{B} \ .$$

From this we can determine the unknown transformation, $\mathbf{T}_{A\to B}^{A}$, in the following way

$$\mathbf{T}_{B\to C}^{B}\left(\mathbf{T}_{A\to C}^{A}\right)^{-1}\mathbf{T}_{A\to B}^{A} = \mathbf{T}_{B\to B}^{B}$$
$$\mathbf{T}_{B\to C}^{B}\left(\mathbf{T}_{A\to C}^{A}\right)^{-1}\mathbf{T}_{A\to B}^{A} = \mathbf{I}$$
$$\mathbf{T}_{A\to B}^{A} = \mathbf{T}_{A\to C}^{A}\left(\mathbf{T}_{B\to C}^{B}\right)^{-1} \ .$$

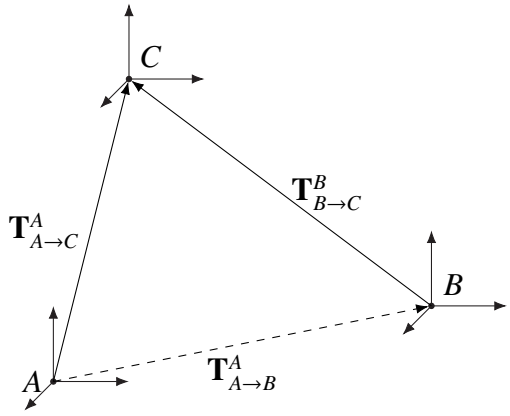

Figure 2.3: Graph of the problem of determining the transformation between two frames given that one frame is described in both base frames.

If we now use the definition of the transformation matrices which were introduced in Eq. (2.1) we can compute the above matrix multiplication to get that

$$\mathbf{T}^{A}_{A\to B} = \begin{bmatrix} \mathbf{R}^{A}_{A\to C} & \mathbf{t}^{A}_{A\to C} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \left(\mathbf{R}^{B}_{B\to C}\right)^{\mathrm{T}} & -\left(\mathbf{R}^{B}_{B\to C}\right)^{\mathrm{T}} \mathbf{t}^{B}_{B\to C} \\ 0 & 1 \end{bmatrix} ,$$

$$\begin{bmatrix} \mathbf{R}^{A}_{A\to B} & \mathbf{t}^{A}_{A\to B} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}^{A}_{A\to C}\left(\mathbf{R}^{B}_{B\to C}\right)^{\mathrm{T}} & \mathbf{t}^{A}_{A\to C} - \mathbf{R}^{A}_{A\to C}\left(\mathbf{R}^{B}_{B\to C}\right)^{\mathrm{T}} \mathbf{t}^{B}_{B\to C} \\ 0 & 1 \end{bmatrix} .$$

In this way the pose of B has been described in the frame of A as desired.

2.1.3 Camera Calibration

When imaging a point in three-dimensional space we can use the pin-hole camera model [26, p.55] to map a point to a pixel position. An example of this can be seen in Fig. 2.4.

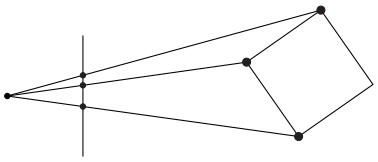

Figure 2.4: Example of imaging the corners of a square according to the pin-hole camera model.

Consider the inverse problem of mapping a pixel position back to the world. Since the projection to the camera plane is not invertible we are not able to determine which three-dimensional point this image position was mapped from, but we can determine which line the three-dimensional point must have existed on.

To perform this inverse mapping we must know the focal length of the camera as well as the pixel position of the image center. If we do not know these parameters we have no way to

determine the line on which the projected point must have existed. Examples of these lines for different values of the focal length and pixel center can be seen in Fig. 2.5.

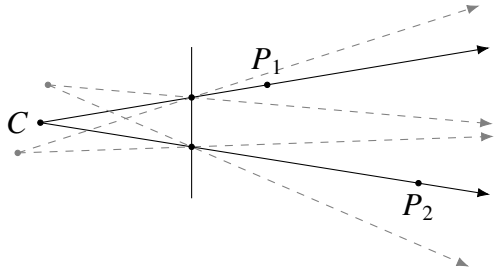

Figure 2.5: The true focal length and pixel center is denoted by C and the imaged points are P_1 and P_2. Only the projection of P_1 and P_2 to the camera plane is known, not the camera parameters which results in many different choices of rays potentially passing through P_1 and P_2.

The pin-hole camera model corresponds to a homogeneous transformation on the following form

$$\begin{bmatrix} x_{\text{pixel}} \\ y_{\text{pixel}} \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \end{bmatrix} \begin{bmatrix} X_{\text{world}} \\ Y_{\text{world}} \\ Z_{\text{world}} \\ 1 \end{bmatrix} .$$

This transformation is usually written as a matrix product in the following way

$$\begin{bmatrix} x_{\text{pixel}} \\ y_{\text{pixel}} \\ 1 \end{bmatrix} \sim \begin{bmatrix} f_x & f_{xy} & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X_{\text{world}} \\ Y_{\text{world}} \\ Z_{\text{world}} \\ 1 \end{bmatrix}$$

$$\sim \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X_{\text{world}} \\ Y_{\text{world}} \\ Z_{\text{world}} \\ 1 \end{bmatrix} .$$

The camera calibration matrix $\mathbf{K}$ is an upper triangular matrix with the focal lengths f_x and f_y and the skew f_{xy} which is 0 for most cameras. The parameters c_x and c_y are the pixel coordinates of the camera center. The matrix $\mathbf{R}$ and vector $\mathbf{t}$ describe the pose of the camera, note that $\mathbf{R}$ is a rotation matrix.

Under the assumption that the pin-hole camera model applies to our camera we can estimate the camera calibration matrix, $\mathbf{K}$. For real cameras the pin-hole camera model is, however, not exact. This can be mitigated by additionally assuming that there exists some non-linear distortion in the imaging process induced by the lens of the camera. We assume that the camera has some radial and tangential distortion described by the equations below

$$x_{\text{d}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + (2p_1 xy + p_2(r^2 + 2x^2)) ,$$
$$y_{\text{d}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + (p_1(r^2 + 2y^2) + 2p_2 xy) .$$

This distortion model is the Brown-Conrady distortion model [1]. The parameters k_1, k_2, and k_3 describe the radial distortion and the parameters p_1 and p_2 desribe the tangential distortion. The true coordinates are x and y while the distorted coordinates we see are x_d and y_d. The variable r is the distance of the true point from the camera center.

The process of estimating this matrix, **K**, and the distortion parameters is called camera calibration or intrinsic camera calibration [26, p.685].

2.1.4 Hand-Eye Calibration

Hand-eye calibration is the process of estimating the relationship between a camera frame and an end-effector frame, i.e. the transformation between the two frames [13, p.3]. In this case the camera either has a fixed position in relation to the environment (hand-to-eye) or to the end-effector (eye-in-hand). Estimating the transformation can be done with the help of calibration objects, such as an ArUco marker or a chess board.

In Fig. 2.6 the setup for eye-in-hand calibration is pictured and the transformations between the different frames have been marked. The transformation between the base frame and the end-effector, **A**, is defined since the structure of the robot and the robots position in the world are known. The transformation between the camera and the object, **B**, is also known since we use the camera to determine the pose of the calibration object in the camera frame. The transformation from the end-effector to the camera, **X**, and the transformation from the base to the object, **Y**, are both unknown.

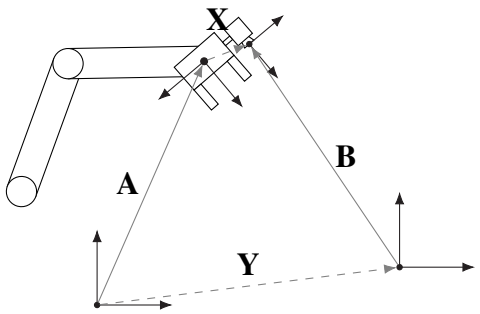

Figure 2.6: The problem setup for eye-in-hand calibration is pictured above. The transformations A, and B are known and the transformations **X** and **Y** are unknown.

In the transformation graph in Fig. 2.6 there are two distinct paths from the camera to the base frame. One of these is the path which passes the end-effector and the other is the path that passes the calibration object. Assuming that the world is consistent, i.e. the transformations that can be derived from these paths are equal since they have the same start and end frame, gives us the following equation [13, p.2]

$$\mathbf{AX} = \mathbf{YB} \ .$$

If we are able to find transformations $\mathbf{X}$ and $\mathbf{Y}$ such that this equality is achieved we know that both $\mathbf{X}$ and $\mathbf{Y}$ have been determined. This tells us where the calibration object is in relation to the base frame but in particular it also tells us where the camera is in relation to the end-effector.

2.1.5 ArUco Markers

In computer vision pose estimation is the problem of estimating the translation and orientation of a rigid body in some coordinate frame. When estimating the pose of an object one usually wants to compare this estimated pose to some ground-truth to evaluate the estimation. More often than not, for arbitrary objects, no ground truth exists. To this end one can use ArUco markers, which are a known pattern with a known size, the pose of which can be determined from a single image.

If an ArUco marker is attached to an object we can compute the pose of the ArUco marker to get a ground truth for what the pose of the object should be. One also needs to record the transformation between the ArUco marker and the object.

An example of an ArUco marker can be seen in Fig. 2.7. Each bit-pattern, the collection of black and white squares, corresponds to a unique id [17]. In the figure we see that the marker has no rotational symmetries which ensures that if we can determine which id this pattern corresponds to we can canonically order the corners and determine its pose.

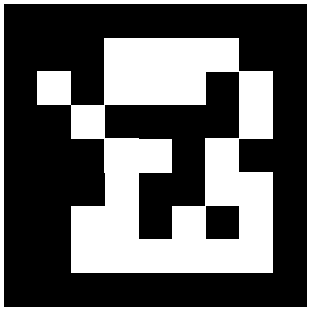

Figure 2.7: An example of an ArUco marker, the bit-pattern has no symmetries to ensure that there is no ambiguity when determining the pose of the marker.

When detecting an ArUco marker we detect the corners of the marker in the image. Since the size of the ArUco marker is assumed to be known we can solve the *Perspective-n-Point-problem*, PnP, to recover the position of the corners of the marker in relation to the camera which took the image [11].

The PnP-problem is the problem of determining where n points lie in the world given how they relate to each other and how they appear in an image [11]. As long as the intrinsic camera parameters are known this problem can be solved. In Fig. 2.8 we can see this setup for an ArUco marker. We know that the imaged corners define the rays that the true position of the corners must exist on, this is sufficient to recover the true positions of the corners.

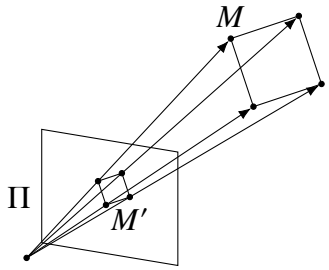

Figure 2.8: A marker M has been imaged in the camera plane Π above. The positions of the corners relative to each other is exactly known and they must exist along the rays that have been marked. From this their position in the world can be determined.

Once the positions of the corners have been determined in the camera frame we can use the fact that the corners are coplanar to describe the orientation of the marker in the camera frame by determining an orthonormal basis that has one vector normal to the plane the corners are defined in.

2.1.6 LiDAR

LiDAR is a type of sensor that can be used to extract length information from the environment. A laser is pointed in a direction and the time it takes for the light to move from the light source, bounce from an object and back to the sensor can be used to calculate how far away the object is [20].

This can be used to map an environment but it can also be used to determine where a set of sensor readings were taken from, given a map. It is particularly useful as a safety feature for avoiding objects such as humans.

2.1.7 Force Sensing

Many robots use some kind of force sensing to interact with the environment with precision. By measuring the forces and torques acting on the end-effector of a robot arm, a robot can adjust its movements and behaviours to achieve specific goals as well as making the movement more safe, e.g. by preempting a motion if an unexpected force is sensed. This is usable for autonomous robots that have to react without the intervention of humans, for example the Mars Rover uses force sensing for accurate performance where responsive human control is impossible [22].

Generally force sensing will be affected by different factors that can make it inaccurate. The weight of the end-effector might have to be accounted for or it will compromise the accuracy of the force-torque sensor. Since sensors are affected by noise it can be hard to get an accurate reading. This can be fixed by filtering. [14]

2.2 Navigation

2.2.1 Mapping & Offline Planning

Many robots are able to navigate autonomously to a goal location if one is specified. This is most often achieved by introducing a map of the environment. There are many different ways to construct a map and a multitude of ways to represent it. A map can be represented as continuous shapes via equations, but it can also be represented with discrete shapes, e.g. an occupancy grid [18].

When a map is described with the help of an occupancy grid, the world is divided into a discrete set of cells. Each cell can have one of two states, occupied or not, which helps the robot figure out where it can move as well as determine where it is [23, ch.9].

A map enables the robot to both plan a path to the goal before the robot starts moving as well as recognize when the goal has been reached. Fig. 2.9 shows an example of a map with a robot that needs to plan a path to reach a goal within the environment. Prior to movement, the path is planned while taking into account the size of the robot and any obstacles present in the map. As the robot navigates through the environment, it must determine its position through sensor readings. This localization process helps the robot identify when it has reached its goal. [23, ch.10]

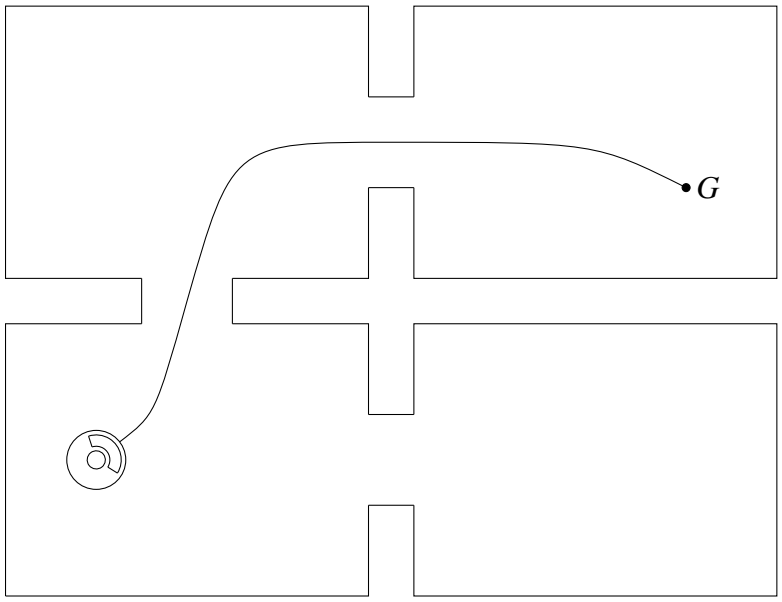

Figure 2.9: A set of rooms is represented in the map above. A path from a robot in the environment to a goal, ***G***, is also drawn.

2.2.2 Online Planning

While traversing a pre-planned path sensor readings can reveal unmapped objects. These obstacles can then be introduced to the map and taken into account when navigating in the future. If the obstacle makes the pre-planned path invalid, suitable recovery behaviours can be used. Recovery behaviours are different techniques to reassess and handle situations as they appear [10].

In Fig. 2.10 an initial path to the goal has been planned which intersects with an unmapped object. The replanned path which avoids the object is marked as a dashed line.

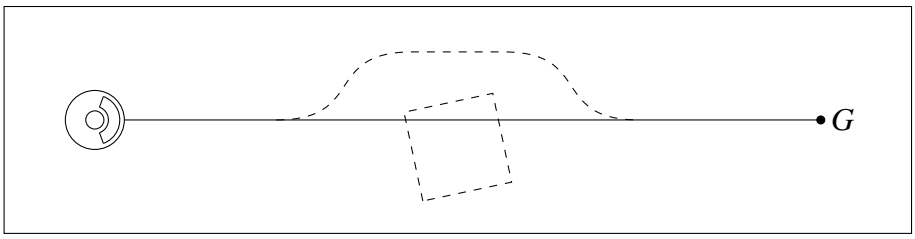

Figure 2.10: Example of an initial path which needs to be replanned when the unmapped obstacle is perceived.

LiDARs are commonly used for sensing the environment, although it is also possible to use cameras or radars for this purpose.

2.3 Actuation

2.3.1 Kinematics

When talking about kinematics in robotics one usually talks about the kinematics of robot arms. These consists of multiple links. Each joint of an arm is controlled by a motor which controls the angle of the joint.

There are two ways of looking at a robot arm, an outer and an inner perspective. The outer viewpoint looks at the pose of each link of the arm in the world. This viewpoint is important for determining if the arm will collide with itself or with the environment. The inner viewpoint looks at the pose of each link of the arm from inside the robot, that is to say, the state of the arm can be described by the angle of each joint. The inner viewpoint is used to control the robot since when controlling the robot we control the joint angles. Forwards and inverse kinematics is the bridge between these perspectives. [27, p.9].

Forward Kinematics

A robot arm consists of multiple links that are serially connected. The position of these links are related to each other by coordinate transformations that are parameterized by the angle at that joint. An example of an arm with the corresponding angle for each joint can be seen in Fig. 2.11. The transformations between the links can be expressed with homogeneous coordinate transformations defined in section 2.1.2.

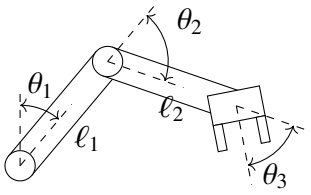

Figure 2.11: A robot arm which has joint angles θ_1, θ_2 and θ_3 is depicted above. The length of the first link is ℓ_1 and the length of the second link is ℓ_2.

The robot arm has some base frame which the first link is defined in relation to. This can then be used to determine where each link will be located for some specific joint angles with these homogeneous transformations. [27, p.26].

The purpose of forward kinematics can most easily be understood with the help of an example. Consider an arm with two links as in Fig. 2.11. To determine where the end-effector is located in the base frame given some joint angles we can consider the transformation between each link separately. In Fig. 2.12 the problem of determining where the frame of the first link is located given some angle can be seen.

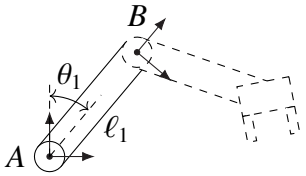

Figure 2.12: The transformation from frame $\boldsymbol{B}$ to frame $\boldsymbol{A}$ needs to be determined.

The origin of frame $\boldsymbol{B}$ can be described in frame $\boldsymbol{A}$ with the following formula

$$O_A^B = \begin{bmatrix} \ell_1 \cos\theta_1 \\ \ell_1 \sin\theta_1 \end{bmatrix} .$$

The new basis vectors of frame $\boldsymbol{B}$ can be described in frame $\boldsymbol{A}$ in the following way

$$\hat{\mathbf{x}}_A^B = \begin{bmatrix} \sin\theta_1 \\ \cos\theta_1 \end{bmatrix} , \quad \hat{\mathbf{y}}_A^B = \begin{bmatrix} -\cos\theta_1 \\ \sin\theta_1 \end{bmatrix} .$$

Once the origin and axes of frame $\boldsymbol{B}$ have been described in the coordinates of frame $\boldsymbol{A}$ we can assemble the homogeneous transformation matrix which computes the change of frame from frame $\boldsymbol{B}$ to frame $\boldsymbol{A}$

$$\mathbf{T}_{A\to B}^{A} = \begin{bmatrix} \begin{bmatrix} \hat{\mathbf{x}}_A^B & \hat{\mathbf{y}}_A^B \end{bmatrix} & O_A^B \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \sin\theta_1 & \cos\theta_1 & \ell_1\cos\theta_1 \\ -\cos\theta_1 & \sin\theta_1 & \ell_1\sin\theta_1 \\ 0 & 0 & 1 \end{bmatrix} .$$

We are interested in determining the pose of the end-effector in frame $\boldsymbol{A}$. The pose of the second link in frame $\boldsymbol{B}$ can easily be determined in a very similar way to the previous transformation. In Fig. 2.13 this problem is presented.

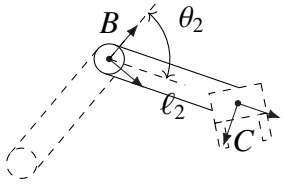

Figure 2.13: Here the homogeneous transformation from frame $\boldsymbol{C}$ to frame $\boldsymbol{B}$ needs to be determined.

The transformation matrix from frame $\boldsymbol{C}$ to frame $\boldsymbol{B}$ is identical to the transformation matrix from frame $\boldsymbol{B}$ to frame $\boldsymbol{A}$. Since homogeneous transformations can be composed to recover a new homogeneous transformation we can see that the transformation from frame $\boldsymbol{C}$ to $\boldsymbol{A}$ becomes the following

$$\mathbf{T}_{A\to C}^{A} = \mathbf{T}_{A\to B}^{A}\mathbf{T}_{B\to C}^{B} = \begin{bmatrix} \sin\theta_1 & \cos\theta_1 & \ell_1\cos\theta_1 \\ -\cos\theta_1 & \sin\theta_1 & \ell_1\sin\theta_1 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \sin\theta_2 & \cos\theta_2 & \ell_2\cos\theta_2 \\ -\cos\theta_2 & \sin\theta_2 & \ell_2\sin\theta_2 \\ 0 & 0 & 1 \end{bmatrix} .$$

The transformation matrix for the last link, the end-effector, thus has the following form

$$\mathbf{T}_{C\to D}^{C} = \begin{bmatrix} \sin\theta_3 & \cos\theta_3 & 0 \\ -\cos\theta_3 & \sin\theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} ,$$

where the frame of the end-effector is frame $\boldsymbol{D}$. These transformation matrices completely describe the kinematics of this robot arm. This can be extended in the same way to arms with multiple links and higher dimensions.

Inverse Kinematics

Inverse kinematics is the problem of determining the joint angles of the arm given a pose. When controlling the arm we are not specifically interested in controlling where each link should end up, we are mainly interested in positioning the end-effector correctly. Therefore inverse kinematics calculates what joint angles an arm must have for the end-effector to end up at a given pose in the world.

The number of joints determine how many degrees of freedom (DOF) the arm has. The pose of the end-effector has three DOF in two dimensions and six DOF in three dimensions. If an arm has less than or the same amount of DOF as the pose there will either be finitely many or no solutions to the problem. If the arm instead has more DOF than the pose there can be any number of solutions to the problem, finitely many, inifinitely many or none. [27, p.27].

This can be seen in Fig. 2.14 where two solutions are presented for an arm with 3 DOF in two dimensions.

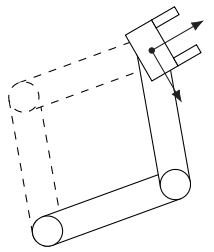

Figure 2.14: The same pose of the end-effector has been achieved with different joint angles.

2.3.2 Control

In robotics, control is essential for accurate robot motion. There are two primary types of control methods: feedback and feedforward. They are used for the same goal but in different circumstances depending on what the situation looks like.

Feedback Control

Feedback control is a control method that compares the output variables of a process to some desired values to create a control signal. Such variables could be angles, velocities and accelerations.

One of the most used feedback control methods is proportional-integral-derivative (PID) control. The PID control bases the control signal on the error between the current and the desired state. The proportional term adjusts the control signal based on the current error, the integral term adjusts the control signal based on the accumulated error over time and the derivative term adjusts the control signal based on the rate of change of the error. [27, p.139].

Feedforward Control

Feedforward is a control method that uses the modelled dynamics of a system to predict how it will behave. The model takes information such as mass, lengths and friction and create a control signal using the following equation

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \tau$$

where $\mathbf{q}$ is the vector of joint positions, $\dot{\mathbf{q}}$ is the vector of joint velocities, $\ddot{\mathbf{q}}$ is the vector of joint accelerations, $\mathbf{M}(\mathbf{q})$ is the joint-space inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is the Coriolis and centrifugal matrix, $\mathbf{G}(\mathbf{q})$ is the gravitational torque vector, and $\boldsymbol{\tau}$ is the vector of joint torques that should be applied at each joint. [27, p.141].

2.3.3 Controllers

The previous section describes specific ways to control a dynamic process. This section instead describes how a process can be controlled, which space we control the process in and what properties the controller might have.

Joint/Linear

The arm can be controller both in joint space and in Cartesian space. Controlling the arm by specifying what joint values it needs to reach is the easiest option. As we control the arm we need to make sure that there are no self collisions or any collisions with the environment.

If a path is constructed in joint space we only know where the arm will end up, not where it will be while it moves there [2]. A path can also be constructed in Cartesian space, then we will know exactly where the end-effector will go, but how the rest of the arm will move as it reaches the goal is unknown.

Stiff/Compliant

In traditional control we are usually interested in controlling a state to some reference value as quickly and safely as possible. In robotics control is only a tool for performing tasks like picking an object or assembling a product.

For industrial robotics it is natural that the control should be precise since many interlocking processes need to complete their tasks in unison. This means that the controller which moves the robot needs to be very stiff, i.e., the controller should have a low error threshold to ensure that it always stays at the reference point.

For autonomous robots in the real world there is no need to be so precise since objects that might need to be manipulated cannot be expected to exist exactly where we expect them to be. For autonomous robots it is also not important that the arm follows a trajectory exactly, it is just important that the trajectory is followed closely enough. This means that we can use a compliant controller which does not have large gains.

The advantage of using compliant control over stiff control is that stiff control follows the reference trajectory with no regard for what might be in the way. For example a human might be by the door you want to open, or the button might not be where the robot expected it to be. With compliant control following the reference is not as important and thus if something

appears in the way of the arm when it's moving the consequences will not be as severe as if stiff control was used [5].

2.4 Ontologies and World Models

Ontologies and world models are tools to organize the world into semantically meaningful objects and their relationships. A door does not exist in a vacuum for example, it connects two rooms to each other. By specifying how objects in the world relate to each other and what properties they have they gain a semantic meaning, e.g. a room can be uniquely characterized by how it is related to other objects in the world.

A world model can thus be defined as a graph with relations between objects in the world. For many applications this is not a sufficient model, one should also keep track of the properties and types of the objects and their relations in the world model. A door is very distinct from a robot arm for example. For this reason we construct a ontology [19] of properties, types and relations which can be seen in Fig. 2.15.

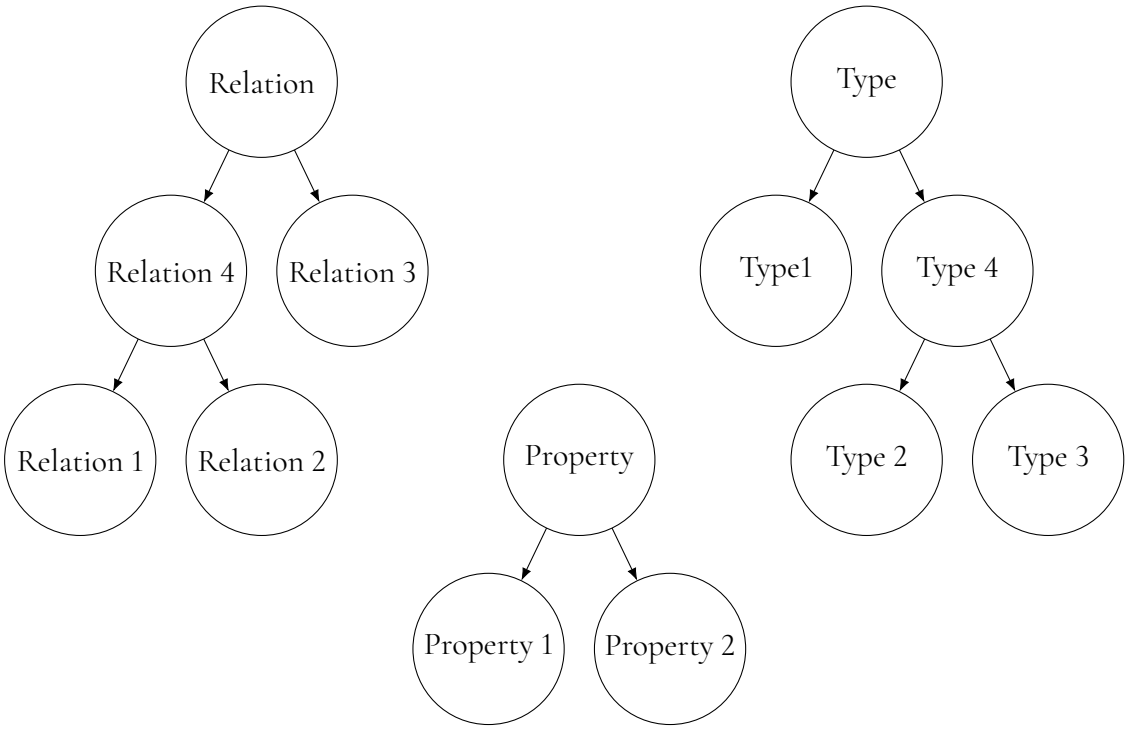

Figure 2.15: An example of an ontology.

A world model enables a robot to semantically interpret the world around it and infer objectives and how skills should be executed [16]. In the world model one needs to specify the robot and everything that the robot needs to know to operate correctly.

An example of a type is *DoorButton* and an example of a property could be *IsStationary*. Finally an example of a relation could be a contain relation, i.e. *object is contained in room*.

2.5 Robot Skills

A robot is used to perform or assist with some kind of task. From a higher level perspective many tasks have overlapping aspects. For example a pick task and a place task both require the robot to detect a suitable location to move the arm to, move the arm to that location and the gripper needs to be actuated.

The breakdown of tasks into independent parts that can be repurposed can be formalized with the help of skills. A skill can be defined to be an action which applies a state transition to the world, i.e. the state of the world is somehow changed by the robot executing the skill. This action can be generalized to different hardware, i.e. a pick skill can be uniquely described without specifying of how it should be implemented on a robot.

Two different types of skills that will be used throughout this report, compound skills and primitive skills. A compound skill is a skill which executes other skills while a primitive skill is more simple and can deal with the specifics of the robot.

A skill can not be executed at any time. For example a pick skill can only (or rather should only) be executed if the gripper is empty, if there is an object to pick and so on. This can be translated into what we refer to as pre-conditions. These need to be satisfied before the skill can be applied.

Additionally some conditions need to hold while the skill is being executed, these can be translated into hold-conditions. For a pick skill the object should not be moving in an unexpected manner while the robot is grasping it.

Similarly to pre-conditions we can formulate post-conditions. These are conditions that are expected to be true if the skill has been executed correctly. After an object has been successfully picked we expect it to be located in the gripper for example.

For any skill an arbitrary amount of conditions can be specified. Some pre-conditions of a pick skill could be the following:

- The position of the object is known.
- The surface of the object has a large enough friction coefficient.
- The object is close enough to the robot for the arm to reach it.
- The object has an orientation which makes it graspable.

Many more subtle pre-conditions can be specified. For any reasonable implementation of skills we make some reasonable assumptions to avoid specifying an endless amount of increasingly obscure pre-conditions.

2.5.1 Coordination of Skills

The execution of skills can be coordinated with the help of a behaviour tree (BT). BTs are structures that help formalize how a skill should be executed. With BTs one can choose if skills should be executed in parallel or sequentially, if skills should be restarted if they fail and so on [7, p.9].

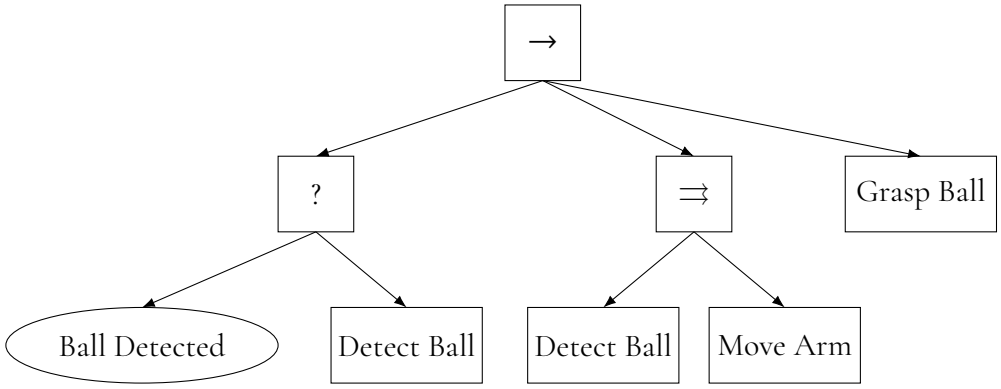

Figure 2.16: A pick skill realized with the help of a behaviour tree.

An example of a compound skill which makes a robot pick up a ball can be seen in Fig. 2.16. The → symbol denotes that all the children of that node should be executed in that order. The question mark denotes that if the condition is true (Ball detected) the skill is executed, otherwise the other child of that node should be executed. The ⇉ symbol denotes that all children of that node should be executed in parallel.

This particular pick skill starts by detecting the position of the ball if it is not known. It then moves the arm to the ball while continuously detecting where the ball is. Finally the ball is grasped.

By constructing primitive skills and other compound skills and then coordinating them with the help of behaviour trees one can easily build compound skills from smaller building blocks in a natural way.

2.5.2 SkiROS

SkiROS is a package built on top of ROS that enables one to program robot-agnostic skills that make use of a world model [21]. This package implements a lot of base functionality which needs to be present when coordinating skills.

SkiROS implements a world model which can be interfaced with. We can also extract and infer information from the world model. Skills can be written as plain python code and behaviour trees can easily be utilized to coordinate the execution of skills.

Chapter 3

Implementation

This chapter will cover what we developed to enable Heron to navigate in a building. To ensure that each part of the project works separately and can be combined to create the final skill we have evaluated each part separately.

3.1 Perception

In this section we collect everything related to Herons perception. We have performed camera calibration, pose estimation, door state classification as well as filtering of Heron's force-torque sensor.

3.1.1 Camera Calibration

To perform precise ArUco detection we need accurate calibration parameters. The cameras we have available are the RealSense RGBD cameras, L515 and D435. These cameras keep track of their own calibration parameters but the true values might change over time, therefore it might be necessary to verify them.

Camera calibration is performed with a calibration pattern, most often a checkerboard. The three-dimensional relationship between the intersections of the squares is exactly known since they are coplanar and equidistant.

The intersections between the squares in the checkerboard are localized in every image, this

together with the fact that the three-dimensional relationship between the intersections is known gives us enough information to compute the calibration parameters.

3.1.2 ArUco Marker Pose Estimation

Pose Estimation

Detecting an ArUco marker and determining the pose of the ArUco marker in the camera frame can be done with the help of pre-written functions in OpenCV that use the attributes of a marker to find its pose [17]. We specifically want to determine the pose of an object which has some known transformation to one or several ArUco markers. This is a fairly simple problem with the derivation from Section 2.1.2. The problem setup we have can be seen in Fig. 3.1 where $\boldsymbol{O}$ is the frame of the object, $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are the ArUco markers, $\boldsymbol{B}$ is the object base frame, and $\boldsymbol{C}$ is the camera frame.

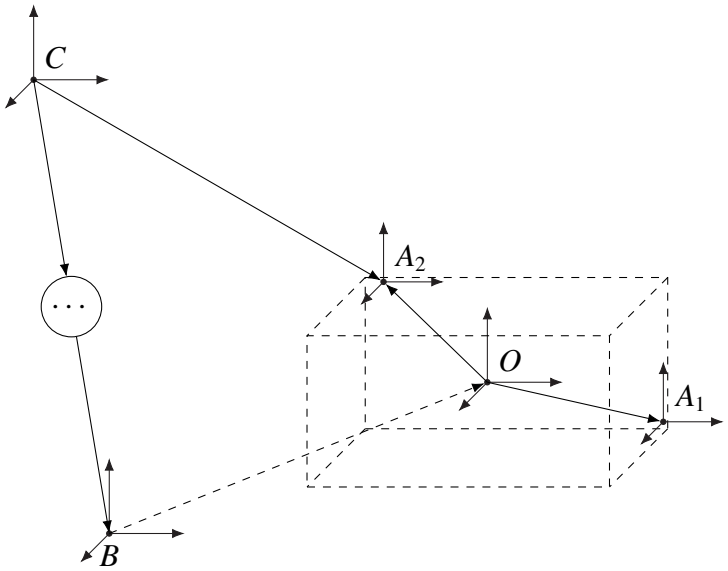

Figure 3.1: The pose of the object with frame $\boldsymbol{O}$ needs to be determined in the object base frame $\boldsymbol{B}$. The pose of each ArUco marker $\boldsymbol{A}_i$ in the object frame is known and the poses of some ArUco markers have been determined in the camera frame $\boldsymbol{C}$.

The coordinate transformations between the camera and some, or all, of the ArUco markers on the object have been determined by solving the PnP-problem. The coordinate transformation between the camera and the object base frame is known and the constant transformation between the object and the ArUco markers is known.

To simplify the problem we start by computing the transformation from the object base frame to every ArUco marker that has been determined in the camera frame. This is possible because all transformations from the ArUco markers to the base frame are known, i.e. a path

must exist in the graph of transformations. To see why this path must exist note that hand-eye calibration has been performed, which determines the position of the camera in the world with respect to the robot.

Once the poses of the ArUco markers have been determined in the base frame the problem in Fig. 3.1 can be simplified to the problem in Fig. 3.2. The transformation from the base frame to ArUco marker i, $\mathbf{T}^W_{W\to A_i}$, is determined by the ArUco detection and the transformation from the object to ArUco marker i, $\mathbf{T}^O_{O\to A_i}$ is fixed and known. Now the transformation from the base frame to the object can easily be determined.

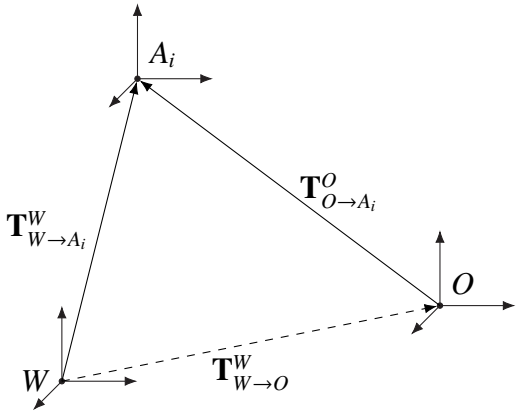

Figure 3.2: Graph of the simplification of the ArUco marker problem.

Note that we have now transformed the problem of determining the pose of the object in the object base frame to the problem in Fig. 2.3. By this restatement of the problem we can determine the pose of the object in the object base frame with the following formula

$$\mathbf{T}^B_{B\to O} = \mathbf{T}^B_{B\to A_i}\left(\mathbf{T}^O_{O\to A_i}\right)^{-1} \quad .$$

For each ArUco marker a separate estimate of the object pose is computed which can then be averaged to get a better estimate of the object pose. The pseudo-code for determining the pose of the object in the base frame can be seen in Algorithm 1.

Algorithm 1 Pseudocode for ArUco marker detection.

procedure ArUcoDetection(I, O) ▷ I, image; O, object to localize
  $M_O \leftarrow$ all ArUco markers associated with O
  $M_e \leftarrow$ find all markers in I and their corners. Make sure the marker ids are in M_O
  for every marker m in M_e **do**
    $\mathbf{R}^O_{O\to A}, \mathbf{t}^O_{O\to A} \leftarrow$ get the known pose of the current marker in the object frame
    Undistort the pixel position of the corners of m with the camera parameters
    $\hat{\mathbf{R}}^C_{C\to A}, \hat{\mathbf{t}}^C_{C\to A} \leftarrow$ recover pose of marker by solving the PnP-problem
    $\hat{\mathbf{R}}^B_{B\to A}, \hat{\mathbf{t}}^B_{B\to A} \leftarrow$ transform the pose of the ArUco marker to the object parent frame
    $\hat{\mathbf{R}}^B_{B\to O} \leftarrow \mathbf{R}^B_{B\to A}\left(\mathbf{R}^O_{O\to A}\right)^{\mathrm{T}}$
    $\hat{\mathbf{t}}^B_{B\to O} \leftarrow \mathbf{t}^B_{B\to A} - \mathbf{R}^B_{B\to A}\left(\mathbf{R}^O_{O\to A}\right)^{\mathrm{T}}\mathbf{t}^O_{O\to A}$
  end for
  compute the average of the estimated pose of the object for every arcuo marker
  return $\hat{\mathbf{R}}^B_{B\to O}, \hat{\mathbf{t}}^B_{B\to O}$
end procedure

Pose Estimation Evaluation

Evaluating the accuracy of the pose estimation requires knowing the ground-truth, i.e. the true physical position and orientation of the ArUco marker and the object it is attached to. In the real world we do not know the true physical pose so in absence of this ground-truth one could instead evaluate the consistency of the pose estimation.

This evaluation can be done by keeping an object to be detected fixed and taking several images. This gives a set of estimates of the true pose $\{\hat{\mathbf{T}}_i\}_i$ and while we are not able to compute the mean deviation from the true pose, $\mathbf{T}$, we can instead compute the standard deviation of the estimated poses. If the pose estimation is correct and unbiased the standard deviation will be small.

Computing the standard deviation gives in each coordinate σ_x, σ_y and σ_z. We can then compute the final error measure in the following way

$$\sigma = \sqrt{\sigma_x^2 + \sigma_y^2 + \sigma_z^2} \ .$$

By moving the camera in different ways, or indeed not moving it at all, we can evaluate the pose estimation for several use cases. One experiment is to keep the camera stationary and estimate the pose for many images. Another is to keep the distance from the object constant while moving the camera around the object.

In the first experiment we evaluate how noise in the image might affect the estimate as well as how consistent it is. In the second experiment we evaluate both how well the translation is estimated as well as the orientation.

If the standard deviation of the estimated poses is small for both of these experiments we can conclude that the pose estimation algorithm gives the correct answer unless there is some

constant offset. This constant offset would then be zero if the hand-eye calibration has been performed correctly.

3.1.3 Door State Classification Using LiDAR

The LiDAR measurements of the robot can be leveraged to determine if a door is open or closed which will help us detect if the robot can go through the door or not.

The position of the door is expected to be known, from this we can specify two bounding boxes in the map. The first and smaller bounding box is the region of the map within which we expect LiDAR measurements if the door is closed. The second and larger bounding box is the region of the map within which we always expect to see LiDAR measurements if the door is present. An example of the two bounding boxes for a door can be seen in Fig. 3.3.

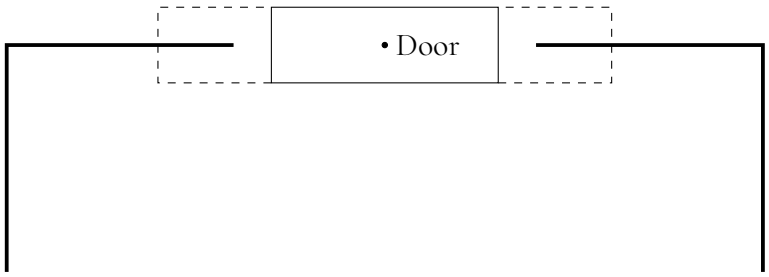

Figure 3.3: The location of the door has been specified as well as two bounding boxes. The smaller one, with solid lines, denotes where we expect the door to be when it is closed. The larger bounding box, with dashed lines, denotes where we expect LiDAR readings to always exist even if the door is open.

By using the larger bounding box which should include the door frame we can tell the difference between not being able to see the door and seeing an open door.

By transforming the LiDAR measurements to coordinates in the map we can verify that the door frame is visible, then we can determine if the door is open or not by choosing a threshold amount of points for each state.

3.1.4 Force Sensing

The force sensing signal includes both noise as well as an offset. These issues can be mitigated by filtering the signal in an appropriate way. The methods that have been chosen for filtering the signal are the following

- Computing a moving average.
- Offsetting the force by a constant vector.

- Turning force sensing on and off completely.

The moving average was computed by averaging the last n sensor readings. When choosing the size of the window, n, one should take the update rate of the sensor signal into account.

A more general filtering technique can be applied by investigating what the force-torque signal looks like in the time-domain as well as in the frequency domain. By investigating this we can choose a more appropriate filtering technique if a moving average is not sufficient.

The offset of the force-torque readings was computed by assuming that the sensor is not experiencing any force other than gravity and a bias. The signal was recorded for a short time-period and the average of the forces and torques during that time was used as the offset.

By turning the force sensing off the controller can not react to forces experienced by the gripper which turns it into a stiff controller. This is not desirable when interacting with objects that could break or similar but can help the controller reach its goal by removing the bias from the force-torque sensor.

Under the assumption that turning off the force sensing helps the controller move the arm correctly a natural extension is to scale the force up or down instead of turning it off completely. By scaling the force sent to the controller it becomes more or less sensitive to external forces. For some tasks the gripper should be more delicate and for other tasks the gripper does not have to worry about breaking anything.

When pressing the button to open the door the robot should feel for a sufficient force before stopping the button press. The force-torque sensor might include a bias which changes over time. This would make choosing the correct force threshold for when the button should be considered pressed difficult. Instead we choose a large initial force goal when we consider the button pressed. This limit is then slowly decreased over time. This makes the initial choice of force goal less important.

3.2 Navigation

3.2.1 Driving Implementation

The low-level implementation of the navigation is taken care of by an action server that plans a path to a specified goal location and drives the robot to the goal. We interface with this server by sending a goal and we can continually verify that the robot is navigating to the goal and we can preempt the goal if desired.

Two recovery behaviours were implemented to handle navigation failures. The primary behaviour is to let the robot try to re-plan a path from where it stands. If the object is unavoidable from the current position of the robot or if it got stuck, a secondary recovery behavior is used. This one continuously saves old positions and allows the robot to go back to the last saved position and re-plan from there.

The navigation server is capable of dealing with unanticipated objects that appear in the robot's path. This corresponds to online planning. The two previous recovery behaviours are more similar to offline planning that use new information. Initially Heron plans a path offline which reaches the desired goal, then it tries to follow that path, if the path can not be followed for any reason Heron uses the online planner.

3.2.2 Navigation Evaluation

Similarly to evaluating the pose estimation in Section 3.1.2 evaluating Heron's navigation ideally requires comparing the estimated position to some known ground-truth. Again, in the real world, we do not know the true physical location of Heron and can not verify that Heron always navigates to the same position if given an identical goal each time.

Under the assumption that the previous algorithm for pose estimation is correct (or at least consistent) the consistency of the navigation can be evaluated by leveraging the pose estimation. An experiment that can be performed is then to place an ArUco marker at some location, drive Heron to that location such that the marker can be seen without moving the arm.

The estimated pose of the ArUco marker can then be saved and the standard deviation can be estimated from several of these poses. Similarly to the pose estimation we can conclude that the navigation is consistent if this standard deviation is small. When choosing an initial location for Heron and a marker location it is important that these locations are quite far from each other. This should ensure that the error in the final position is uncorrelated to the error in the initial position.

3.3 Actuation

3.3.1 Actuation Implementation

Two different controllers were used to control the arm movement. A stiff controller which controls the arm in joint space and a compliant controller which controls the arm in Cartesian space. These controllers already existed and were merely wrapped such that they could be used as skills in SkiROS.

Both controllers were necessary to implement as skills. The joint controller is able to move the arm large distances but is stiff and therefore not suitable for pressing the button. The compliant controller is not suitable for moving the end-effector large distances but is suitable for pressing the button, additionally the controller only moves the end-effector from the starting pose to the ending pose along a line, obstacles are not taken into account when producing this line. This is why both of these controllers needed to be used rather than just the compliant controller.

3.3.2 Actuation Evaluation

The evaluation of the accuracy of the arm movement is done similarly to evaluating the pose estimation in Section 3.1.2 and the navigation evaluation in Section 3.2.2. Once again the ground-truth is unknown and is replaced with an ArUco marker we detect.

By using the ArUco detection an experiment can be performed where the arm is moved from an initial location to a location where ArUco detection is possible and then move back.

The estimated poses can be saved and the standard deviation be calculated. A small standard deviation translates to a consistent actuation.

The same experiments where performed for both controllers

3.4 World Model

The world model that was used to run the skills only keeps track of the structure of the building Heron needs to navigate in. This is not a strict limitation since objects that are needed to accomplish additional skills can easily be added to the rooms they are present in.

The ontology which was used to specify objects and relations can be seen in Fig. A.1 and Fig. A.2 in Section A.1 in the Appendix. An example of a small part of the specified world model can be seen in Fig. 3.4. Note that the location that is associated with the door and the one that is associated with the button do not have to be same one.

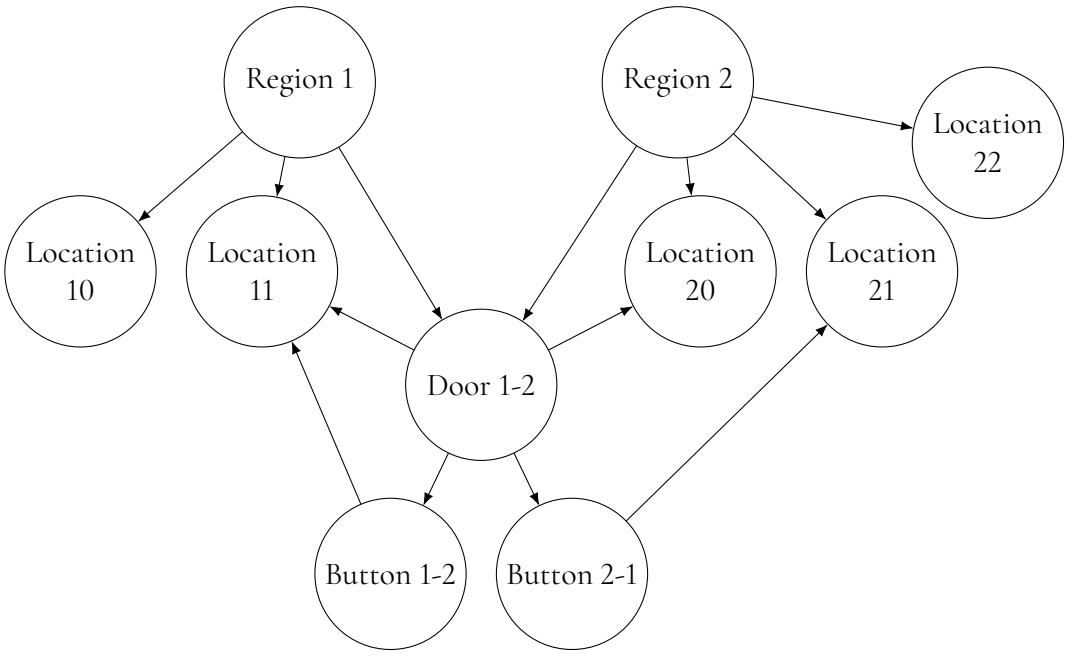

Figure 3.4: A part of the world model.

The relation between a door and a region is a hasDoor relation while the relation between a location and a door is a contain relation. Each button has a PositionKnown property which keeps track of whether its location in the world has been determined or not.

Every region denotes a room in the building while every location represents a specific set of coordinates in that particular room. A location is thus a set of coordinates Heron can always move to as long as Heron's current location and the target location are in the same room. Doors represent some kind of transition between regions (rooms) that can be passed through with the help of a skill. With the use of a door passing skill Heron can move between locations that are not in the same room but have door connections between them.

3.5 Robot Skills

The implemented robot skills were generally small self-contained skills implemented as compound skills. These skills are the ones described below. Many of the skills use objects in the world model to know how and with what inputs perform the skill.

3.5.1 Button Press

The button press skill expects a button as input. This button has specific ArUco markers related to it. With this information the arm moves, using a joint space controller, to a lookout pose which is specific for each button. There the ArUco markers are detected and the exact pose for the button is determined.

If the button is successfully detected the arm moves with a Cartesian controller to a position straight in front of the button, the pre-press pose. After this the arm moves forward, pressing the button, until a strong enough force is felt. The arm then moves back to the pre-press pose and then the lookout pose.

A simplified version of the skill can be seen in Fig. 3.5

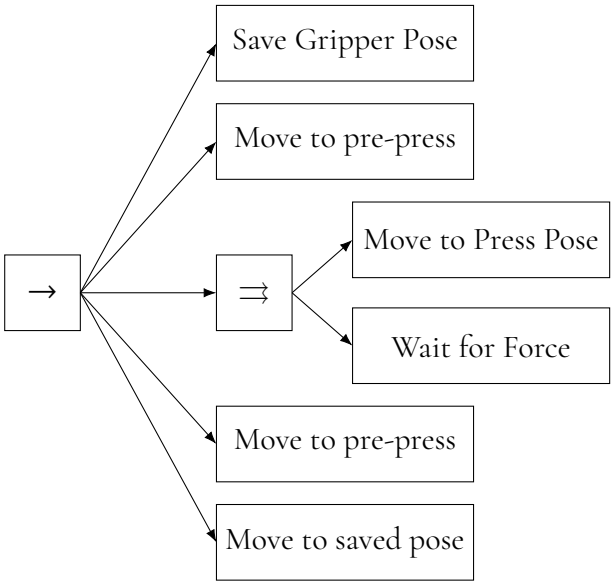

Figure 3.5: A behaviour tree which describes a button pushing skill.

The input which this skill needs is only the button. The preconditions are then that the robot is standing close enough to the button to press it and that the exact coordinates of the button are known. The post-conditions of the button pressing is only that the button has been pressed with a sufficient force. The skill could be used for any kind of button press which is why the correct door being open is not a post-conditions for this skill.

3.5.2 Pass Door

The pass door skill starts with the knowledge of where the robot is and which door it is passing. From this the skill starts with finding the end location on the other side of the door. After this the robot detects the door state while driving towards the goal. The door is assumed to be open when the skill starts execution but if the robot detects that the door is closed while driving towards it the skill will halt execution.

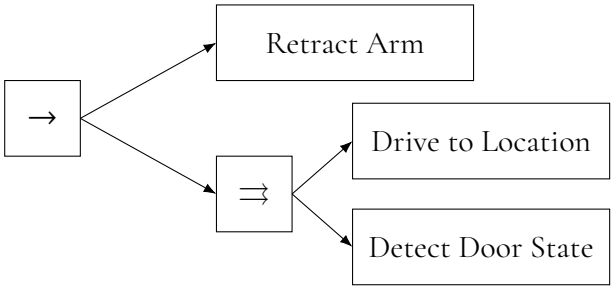

Figure 3.6: A behaviour tree which describes a door passing skill.

The pre-conditions for the pass door skill requires that the robot stands in a room which is related to the door. The target, which is just past the door, can then be inferred from Heron's

current position. The post-condition of this skill is that the target location has been reached.

3.5.3 Path planning

To find a path from one location to another a path planner is built. This uses a breadth-first search to traverse the regions and region transitions, finding the final location and planning a path how to get there. This then returns a list of locations and region transitions that the robot has to pass to get to the goal location.

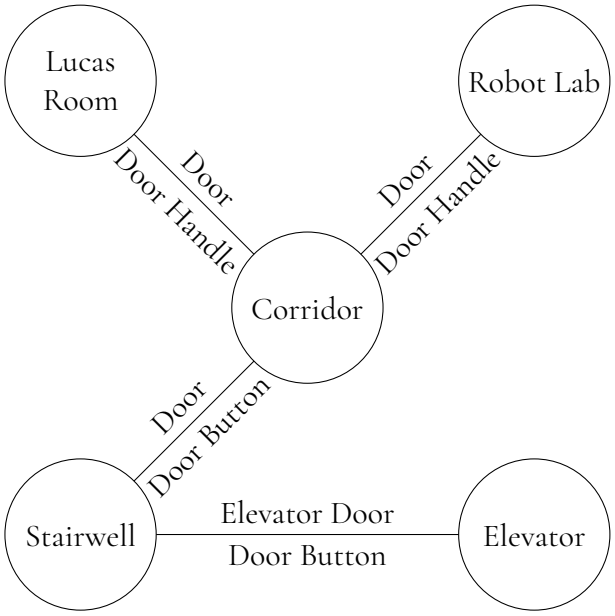

Figure 3.7: A graph representing regions of a floor in the building.

The specific graph structure which we have can be seen in Fig. 3.7. Finding a path between any location in each region is not a very hard problem but by implementing graph search we can very easily find a path between locations in larger buildings.

3.5.4 Final traversing skill

These three skills together with a movement skill are building blocks that can be used to navigate in any building with doors operated by buttons. By using the planning skill the robot can determine which doors and rooms to pass to reach a goal location.

From the planned path we can then assemble a skill sequence which achieves the intended goal of navigating to a specified location. A behaviour tree can not be specified for this skill since a behaviour tree is constructed for each initial and target location the robot should reach. I.e. each time the skill is run the behaviour tree is built dynamically depending on the starting and ending locations.

Chapter 4

Results

4.1 Perception

4.1.1 Camera Calibration

The intrinsic camera parameters have been computed for two RealSense cameras, L515 and D435. The L515 has a resolution of 1280x720 while the D435 has a resolution 848x480. The result for the calibration as well as the factory parameters are presented in Tab. 4.1 and 4.2.

When calibrating the cameras 20 images were taken of the calibration pattern for each experiment. This to ensure that any noise in the estimate of the parameters is kept minimal.

Linear Calibration Parameters	L515		D435	
	Factory	Calibrated	Factory	Calibrated
Focal Length x (f_x)	893.144	904.1047	614.979	612.3398
Focal Length y (f_y)	892.740	903.1287	615.014	612.1060
Pixel Center x (c_x)	653.163	638.2549	430.603	426.4604
Pixel Center y (c_y)	363.806	354.3893	237.271	233.9470

Table 4.1: Linear intrinsic parameters for the two cameras, both the determined parameters and the pre-determined factory parameters are reported for each camera.

Distortion Parameters	L515		D435	
Parameters	Factory	Calibrated	Factory	Calibrated
1st Radial Distortion Coefficient (k_1)	0.1424	0.1359	0.0000	0.1203
2nd Radial Distortion Coefficient (k_2)	-0.4780	-0.4213	0.0000	-0.3019
3rd Radial Distortion Coefficient (k_3)	0.4445	0.3288	0.0000	0.1847
1st Tangential Distortion Coefficient (p_1)	-0.0021	-0.0007	0.0000	0.0002
2nd Tangential Distortion Coefficient (p_2)	-0.0001	-0.0012	0.0000	-0.0044

Table 4.2: Intrinsic distortion parameters according to the Brown-Conrady distortion model. Both the determined parameters and the pre-determined factory parameters are reported for each camera.

4.1.2 Pose Estimation

The consistency of the pose estimation was evaluated with the previously described experiments. The results of this evaluation can be seen in Tab. 4.3.

The static evaluation corresponds to the experiment were Heron's arm was held still and the radial evaluation corresponds to the experiment where the camera was kept at a constant distance from the object with different orientations.

Distance to object (m)		Small Markers		Large Markers	
		σ_p (m)	σ_q	σ_p (m)	σ_q
Static	0.60	0.00333	0.0217	0.000635	0.000532
	1.00	0.0248	0.110	0.00128	0.00117
Radial	0.60	0.0461	0.369	0.0379	0.0172
	1.00	0.0799	0.283	0.359	0.163

Table 4.3: Standard deviation parameters for pose estimation evaluation. The standard deviation of the position is denoted by σ_p while the orientation is denoted by σ_q.

The unit of the standard deviation of the poses is meters. The standard deviation of the orientations does not have a simple interpretation but it is computed from a set of quaternions. The side length of the small markers is 30 millimeters and the side length of the large markers is 174 millimeters.

For the static experiment with the pose estimation 50 images were used to compute the standard deviation while for the radial experiments 10 images were used to compute the standard deviation.

4.1.3 Door State Classification

The LiDAR signal can include noise which makes it so that the door might be detected as closed for a single LiDAR reading. To stop the door detection skill from preempting the navigation skill early a threshold was added. This threshold required that the door must have been detected as closed for at least n LiDAR readings.

4.1.4 Force sensing

The time domain of the unfiltered force sensing signal can be seen in the left plot of Fig. 4.1 and the right plot of Fig. 4.1 shows the smoothed signal. The filter which additionally offsets the bias is not plotted but this filter purely offsets the signal by the estimated mean of each separate coordinate.

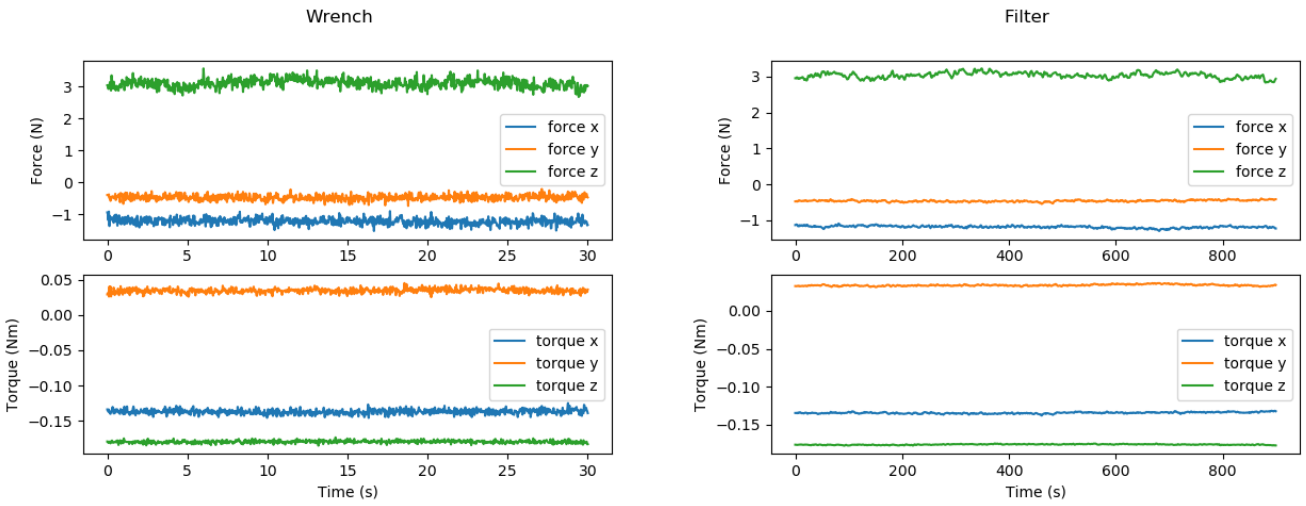

Figure 4.1: Force-Torque signal before and after filtering.

The frequency domain of the same force sensing signal can be seen in Fig. 4.2. The left plot shows the discrete Fourier transform of the unfiltered signal while the right plot shows the discrete Fourier transform of the filtered signal.

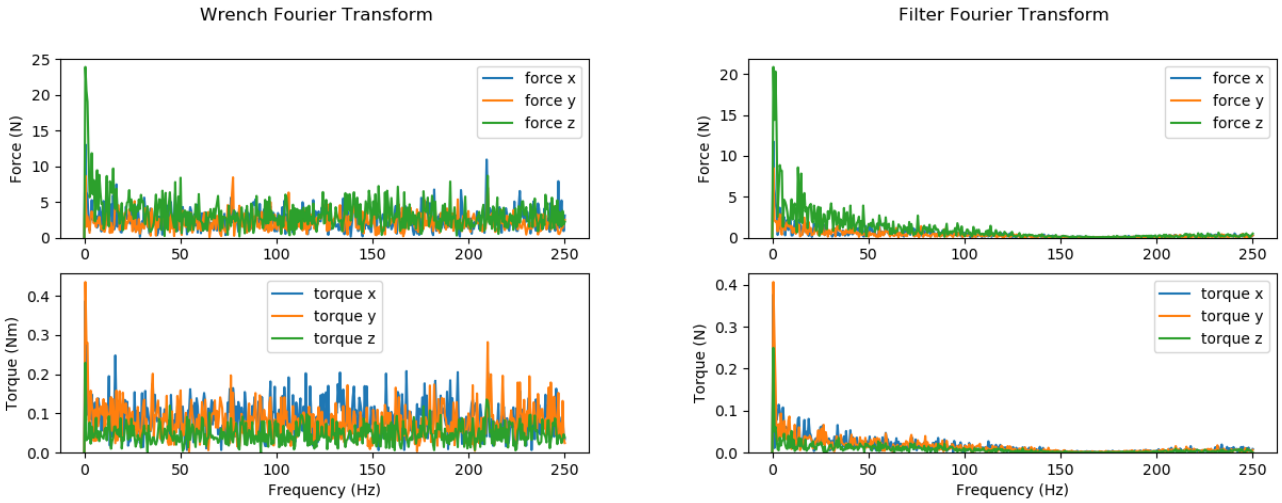

Figure 4.2: Discrete Fourier transform of the force-torque signal before and after filtering.

4.2 Navigation Evaluation

The navigation was also evaluated by estimating the consistency. Only the static experiment was performed without any arm movement. The standard deviation of the translation and orientation of the poses were the following:

$$\sigma_p = 0.158 \text{ (m)} \ , \qquad \sigma_q = 0.0501 \ .$$

For this experiment 20 images were taken to compute these values.

4.3 Actuation Evaluation

To evaluate the actuation an ArUco marker was placed at some location and the arm was moved from some start location to a goal location 20 times per experiment. The results of these experiments can be seen in Tab. 4.4.

Distance to object (m)		Joint Space Control		Cartesian Control	
		σ_p (m)	σ_q	σ_p (m)	σ_q
Static	0.60	0.00177	0.00301	0.00357	0.00478
	1.00	0.00120	0.00261	0.00546	0.00883

Table 4.4: Standard deviation parameters for arm control evaluation.

4.4 Performance of Compound Skills

A demo video which showcases the skills that were implemented for Heron can be seen by scanning the QR code in the Appendix in Fig. A.3. Additionally a demo video which shows that the button pressing skill generalizes to different buttons can be seen by scanning the QR code in the Appendix in Fig. A.4.

4.4.1 Button Pushing

The button pushing has been evaluated on the button in the corridor which opens the door to the stairwell. To evaluate the button pushing fairly, Heron was moved between the button and a location in the corridor far away from the button each time a press was executed. Heron was able to press the button successfully such that the door opened 30 out of 30 times.

4.4.2 Pass Door

The door state detection skill works as well as the navigation so the compound skill which passes a door while making sure it stays open also works.

4.4.3 Path planning

The path planning finds a valid path to the goal location every time. Breadth-first search is used to determine the path which means that the path is guaranteed to pass the least amount of doors. The size of each room is not taken into account when determining the path which means that Heron purely tries to pass as few doors as possible when getting to the goal.

Chapter 5

Discussion

5.1 Consistency Comparison

For each of the parts of the project, evaluations were done. Since a general ground truth did not exist repeated tests were executed. The tests estimated the pose of a static object which gave standard deviations in the position and orientation of the estimated pose. These values have been summarized in Tab. 5.1 and 5.2.

Distance to object (m)		Pose Estimation		Navigation	Actuation	
		Small Markers	Large Markers		Joint	Compliant
Static	0.60	0.00333	0.000635	-	0.00177	0.00357
	1.00	0.0248	0.00128	0.158	0.00120	0.00546
Radial	0.60	0.0461	0.0379	-	-	-
	1.00	0.0799	0.359	-	-	-

Table 5.1: Standard deviation parameters for position evaluation

Distance to object (m)		Pose Estimation		Navigation	Actuation	
		Small Markers	Large Markers		Joint	Compliant
Static	0.60	0.0217	0.000532	-	0.00301	0.00478
	1.00	0.110	0.00117	0.0501	0.00261	0.00883
Radial	0.60	0.369	0.0172	-	-	-
	1.00	0.283	0.163	-	-	-

Table 5.2: Standard deviation parameters for orientation evaluation

The pose estimation was evaluated both with small markers and large markers. For any distance from the object the larger markers gave a more consistent estimate which lets us conclude that if the markers appear larger in the image the pose estimation is more consistent.

Due to this, the experiments which evaluated the navigation and the actuation were performed with large markers. The standard deviation of the pose for these experiments is expected to be at least larger than the standard deviation of the static experiments with large markers. This under the assumption that the same amount or fewer samples are collected which ensures that the standard deviations can be compared.

The standard deviations that were determined for the navigation and actuation evaluation are all larger than the corresponding static pose estimation for large markers, except for the evaluation of the joint space controller at a distance of 1 meter. Even though this standard deviation is smaller than we expect it is very close to the expected lower bound. This could be explained by the fact that only 20 repetitions were performed of these experiments, i.e. this could have been pure chance.

All of the standard deviations are small compared to the size of the motion, both in position and orientation. This is good since it means that the motions are fairly consistent.

The largest standard deviation was reached by radial pose estimation from a distance of 1m. This standard deviation is unusually large compared to the value for the smaller markers. This might be due to the fact that OpenCV sometimes reverses the z-axis of the determined ArUco marker if the ArUco marker is too skewed in the image. This would then lead to a considerably worse estimate which could contribute to a worse standard deviation.

5.2 Perception

5.2.1 Camera Calibration

The camera parameters that were determined are quite close to the factory parameters which are saved on the camera. Under the assumption that the factory parameters were correct for that camera when it was manufactured it is reasonable that our determined parameters are not exactly equal to these since the parameters might change slowly over time.

When the cameras were calibrated we noticed that the estimate is quite sensitive to noise, it is important that the calibration pattern is clearly visible in every picture and not too skewed. Many images are needed to mitigate the noise that can not be eliminated.

5.2.2 Pose Estimation

From the results one can draw the conclusion that larger ArUco marker gives a lower standard deviation. This means that the size of the ArUco marker matters. It is also clear that the closer the ArUco marker is in the image the lower the standard deviation is. This means that the size of the ArUco marker in the image is the factor which determines how exact the pose estimate is.

The fact that the ArUco marker detection is quite consistent for large ArUco markers does not mean that it is correct. We know that the ArUco marker is consistent from a single view and visual inspection of the data looks correct, but when the camera then moves around incorrect hand eye calibration will lead to a non-constant offset in the determined pose.

5.2.3 Door State Classification

The state of a door can be detected reliably when the door is fully open or fully closed. For any intermediate state when the door is half open or half closed the door state classification algorithm leaves a bit to be desired. When the door is in an intermediate state between open and closed the door state classifier does not make a decision on whether the door is open or closed, only that the state is unknown.

Choosing the right threshold for how many points are sufficient to consider the door to be open or closed is a hard problem made even more difficult in the case when a person might be standing in the bounding box of the door. A person could also stand in front of the door in which case Heron would be able to see the door frame but fewer points in the bounding box of the door.

To solve these issues a more complicated approach could be used to detect the state of the door, the rays from the LiDAR could be replicated and checked if they intersect with the bounding box of the door. This new algorithm would then regard the door as open if sufficiently many rays from the LiDAR intersect with the bounding box without stopping in the bounding box. The door would be regarded as closed if sufficiently many LiDAR signals stop in the bounding box. The door state would then be regarded as unknown if LiDAR readings neither stop nor go through the bounding box.

How to handle an unknown door state is determined by what purpose we are detecting the door for. If we expect the door to be open we might want to interpret an unknown door state as closed and if we expect the door to be closed we might want to interpret and unknown door state as open.

5.2.4 Force sensing

When the force sensing was active Heron tended to feel a “ghost force”. This is either due to the weight of the gripper or a bias in the force-torque sensor. To counteract this the offset

force function was implemented. This way we got a start state where the end-effector did not drift.

The compliant controller reacts to forces that affect the end-effector. When pressing the button this led to an oscillating behaviour where the controller feels a force, moves back from the button and then tries moving towards the button again. To prevent this oscillating behaviour a scaling of the force was introduced which scaled the force down. The reactive behaviour of the controller was then smaller which let Heron press the door button correctly..

When trying to push the button the force sensing was quite unreliable. We could not expect to feel the same force every time the button was pressed. This is either due to the fact that the force sensing is inconsistent or because of the compounding errors from the navigation, perception and manipulation which led to the position where Heron pushed on the button to vary. Or the force required to press the force is inconsistent.

5.3 Navigation

The experiment that was performed shows that the navigation is quite consistent, we can expect Heron to navigate closely to the specified goal. The navigation reaches the desired goal with a standard deviation of 5cm, therefore we are not able to press the button by blindly sending the arm to the same joint values every time. This shows that determining the position of the button really is necessary, which also has the added benefit of making the button pushing skill more general. As long as the general location of the button has been determined Heron can navigate there to find it.

In essence the knowledge of where Heron should drive and look to find the button could be replaced with a searching behaviour. As long as the door and its location exists in the world model some searching strategy could be utilized by driving to the door and looking for the door operating mechanism in its vicinity. This would however only be helpful in buildings that are somehow unknown. If the general position of the door operating mechanism has been determined even once there is no reason not to retain that information. The position of the door already needs to be determined in the world model which means that also finding the general position of the operating mechanism does not require a significant degree of added effort.

When using Heron we were trying to enter the elevator. Using the built in navigation in MIR this worked fine but when trying to navigate ourselves it did not work. This is probably because there are some thresholds which are, for safety, higher than the ones MIR uses to make sure it does not break anything, however the time limit prevented us to look in to it and were therefore not able to implement this in the project.

5.4 Actuation

When moving the arm the two controllers were used for different purposes. The compliant controller is in theory the better choice to use since it is compliant and will not hit anything hard but rather be pushed away and prevent destroying something or hurting someone. However when moving long distances or turning big angles the movement was unnecessarily large or unreliable. Because of this we opted to use the joint controller in these cases and the compliant controller when moving close to objects or when small movements were expected. By doing this we got a reliable combination that did not break anything but did not move in an unexpected way.

It is worth noting that the compliant control had a large settling time, i.e. it took the controller a long time to stop moving. The ArUco marker detection, which was used to evaluate the consistency of the controller, performs worse if the camera moves while the images are captured. To mitigate the noise in the estimate a waiting time of 5 seconds was added between when the controller thought the goal has been reached and and when the images were captured. The controller has a translation threshold of 4cm. This means that if the magnitude of the error in the position of the end-effector is less than 4cm the movement is regarded as completed.

The compliant controller has a small standard deviation when it is allowed to move towards the goal for a long time. But the controller decides that the goal has been reached once the translation error is smaller than 4cm, therefore we can only expect the compliant controller to be able to reliably push buttons the dimensions of which are larger than this length. Lowering the threshold is not an option since the compliant controller reacts to the forces that act on the end-effector, the force signal has a bias which can not be mitigated for any orientation of the gripper with our filtering technique, thus some small offset of the end-effector from the goal pose is to be expected.

5.5 Compound Skills

5.5.1 Button pressing

The compound skill which performs the button pressing is consistent and reliable. The reliability hinges on the fact that the hand-eye calibration parameters are correct which lets Heron press the button with precision. As long as the button is large enough and its pose can be determined with reliable accuracy the button pressing very easily generalizes to buttons other than the three it has been tried on.

When the button is being pressed the compliant controller should, for the moment, stop reacting to external forces. Of course within reasonable limits, we do however expect a force to arise when pressing the button so the behaviour of avoiding forces is undesirable. By using the filtering in the force sensing this is solved by scaling the force down which makes the

compliant controller react less to external forces on the gripper.

5.5.2 Pass Door

Passing the door is reliable but retracting the arm to the home position takes a long time. The arm needs to be in the home position before the platform will move, this together with the time it takes for the navigation stack to plan path through the door makes passing the door before it starts to close impossible. None of these factors are in our control which is why this problem has been ignored.

5.5.3 Path planning

The path planning (which determines which doors the robot needs to pass to reach a goal) returns a path with the least amount of doors, it does not, however, take into account the distance between doors. There might be a shorter path that uses more doors. Passing a door is however time consuming and could fail, this means that it could be desirable to pass as few doors as possible when navigating in a building.

Chapter 6

Conclusions

6.1 Research Questions

The goal of this thesis has been to answer the formulated research questions. In this report we have shown that a door opening skill is possible to describe and implement in practice on an autonomous robot. The building navigation skill which was implemented could also easily be extended to include going through doors which are operated by door handles. This is under the assumption that someone has implemented a working skill which enables a robot to operate a door handle.

6.1.1 Reliability

The research questions which were related to the reliability and consistency of the separate parts of the project are the following

- Can we utilize fiducial markers to reliably detect the button?
- How consistent and precise is the manipulation of the arm?
- How consistent and precise is the robot navigation?
- How reliable is the force sensing?
- How reliable is the button pushing as a whole?

Fiducial makers, in our case ArUco markers, have shown to be reliable and useful. Their use has contributed to a accurate and consistent button pushing.

The manipulation of the arm in both joint space and Cartesian space as well as the navigation of the platform works reliably enough on their own. By implementing a button pressing skill it has also been proven that the combination is reliable.

The force sensing is fitful but filtering it in different ways makes the performance satisfactory.

The button pressing skill as can be seen in the result is reliable for the button it was supposed to open.

6.1.2 Planning

The research question which was related to the planning was: Can Heron itself plan how a door opening skill should be utilized to help it navigate in the environment?

We implemented a skill which does exactly this for Heron. The path can then be inspected and the relevant skills can be assembled. This planning skill works well and generalizes to larger environments without problems.

6.1.3 Door Opening

The research questions which were related to the door opening were the following

- How should a door opening skill be specified?
- What conditions does a door opening skill need to be satisified?
- How can a door opening skill be applied to Heron?

The door opening skill which needed to be specified is the button pushing skill. With behaviour trees the skill has been described on a high level which ensures that it generalizes to any other mobile robot with an arm of some kind.

The input which the button pressing skill needed was only the button all other inputs could then be inferred from the world model, the pre-conditions then ensure that the skill can be applied. The pre-conditions are only relevant for the local information in the world model. The structure of the world model which has no relation to the button is irrelevant for pressing the button and going through the door. The post-conditions then describe the expected state of the world after the skill has been executed, which ensures that the skill was executed properly.

The door opening skill was implemented for Heron with the help of SkiROS. The world model could then be used to efficiently specify the input parameters as well as the pre- and post-conditions.

6.2 Future Work

In the future there are many skills that can utilize the skills we have built and apply appropriate extensions. For example one could solve the problem of entering the elevator with Heron which would effectively make Heron able to navigate in the entire building autonomously rather than just one floor. Heron could then navigate in spaces where humans regularly pass.

When another gripper is installed on Heron it might be possible to operate different opening mechanisms, a door handle for example. This will give Heron the possibility to enter more rooms.

Safety is a big part of autonomous systems working in human spaces. The joint controller is more reliable but it is stiff, it can hurt people. On the other hand the way the compliant controller moves can lead to different problems. A solution to this could be to make the Cartesian movements better or make the joint controller compliant to make it as utilizable and safe as possible.

The movement of the arm does not take its environment into account. This makes it possible for it to plan movements that collide with walls, objects and humans. One could find a way to avoid this. It could be using the LiDAR to define walls, using the depth function of the camera or mounting new sensors on Heron.

Putting up ArUco markers might disturb an environment. To avoid this one could figure out a way to detect poses of door operating mechanisms which do not require markers. However this might lead to different problems, for example worse detection.

Since Heron is an inconsistent robot there is a need for some simulation when is comes to button pressing. The simulation works well when it comes to other parts but since there is no reliable force sensing we can not simulate button pressing. A way to test in simulation is always good to have and a way to simulate button pressing would be useful.

When executing an extensive compound skill something could always go wrong. A door not opening, the button not being pushed hard enough etc. This would lead the entire skill to fail. To avoid this one could implement some kind of recovery behavior in the skill. If the door is perceived as closed try to press the button again. If the door is locked and can not be opened, find some way to communicate this to a human and wait for them to open the door. The options are unlimited and are a way to make Heron even more autonomous.

References

[1] M. Ahmed and A. Farag. Nonmetric calibration of camera lens distortion: differential methods and robust estimation. *IEEE Transactions on Image Processing*, 14(8):1215–1230, 2005.

[2] G. Antonelli, F. Caccavale, and P. Chiacchio. A systematic procedure for the identification of dynamic parameters of robot manipulators. *Robotica*, 17(4):427–435, 1999.

[3] Miguel Arduengo, Carme Torras, and Luis Sentis. Robust and adaptive door operation with a mobile robot. *Intelligent Service Robotics*, 14(3):409–425, may 2021.

[4] Francis Bacon. *Essays*. 1625.

[5] Andrea Calanca, Riccardo Muradore, and Paolo Fiorini. A review of algorithms for compliant control of stiff and fixed-compliance robots. *IEEE/ASME Transactions on Mechatronics*, 21:1–1, 01 2015.

[6] Sachin Chitta, Benjamin Cohen, and Maxim Likhachev. Planning for autonomous door opening with a mobile manipulator. *IEEE International Conference on Robotics and Automation*, 05 2010.

[7] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI*. CRC Press, jul 2018.

[8] Carl D. Crane, III and Joseph Duffy. *Kinematic Analysis of Robot Manipulators*. Cambridge University Press, 1998.

[9] Boston Dynamics. `https://support.bostondynamics.com/s/article/Door-manipulation-with-the-Spot-Arm`.

[10] Thomas Eiband, Christoph Willibald, Isabel Tannert, Bernhard Weber, and Dongheui Lee. Collaborative programming of robotic task decisions and recovery behaviors. *Autonomous Robots*, 47:1–19, 10 2022.

[11] Robert Haralick, Chung-Nan Lee, Karsten Ottenberg, and Michael Nölle. Review and analysis of solutions of the three point perspective pose estimation problem. *International Journal of Computer Vision*, 13:331–356, 12 1994.

[12] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Compute Vision*. Cambridge University Press, 2004.

[13] Radu Horaud and Fadi Dornaika. Hand-eye calibration. *I. J. Robotic Res.*, 14:195–210, 06 1995.

[14] Sang-Hyuk Lee, Young-Loul Kim, and Jae-Bok Song. Torque sensor calibration using virtual load for a manipulator. *International Journal of Precision Engineering and Manufacturing*, 11:219–225, 04 2010.

[15] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[16] Joanna Isabelle Olszewska, Marcos Barreto, Julita Bermejo-Alonso, Joel Carbonera, Abdelghani Chibani, Sandro Fiorini, Paulo Goncalves, Maki Habib, Alaa Khamis, Alberto Olivares, Edison Pignaton de Freitas, Edson Prestes, S. Veera Ragavan, Signe Redfield, Ricardo Sanz, Bruce Spencer, and Howard Li. Ontology for autonomous robotics. In *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 189–194, 2017.

[17] OpenCV. `https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html`.

[18] B.K. Patle, Ganesh Babu L, Anish Pandey, D.R.K. Parhi, and A. Jagadeesh. A review: On path planning strategies for navigation of mobile robot. *Defence Technology*, 15(4):582–606, 2019.

[19] Signe A. Redfield. A review of robotics taxonomies in terms of form and structure. *CoRR*, abs/2101.02659, 2021.

[20] Stephen Reutebuch, Hans-Erik Andersen, and Robert Mcgaughey. Light detection and ranging (lidar): An emerging tool for multiple resource inventory. *Journal of Forestry*, 103:286–292, 09 2005.

[21] Francesco Rovida, Matthew Crosby, Dirk Holz, Athanasios S. Polydoros, Bjarne Großmann, Ronald P. A. Petrick, and Volker Krüger. *SkiROS—A Skill-Based Robot Control Platform on Top of ROS*, pages 121–160. Springer International Publishing, Cham, 2017.

[22] Ethan W. Schaler, James Wisnowski, Yumi Iwashita, Jeffrey A. Edlund, Jacqueline H. Sly, William Raff, Kristopher L. Kriechbaum, Matthew A. Frost, Ryan L. McCormick, and Julie A. Townsend. Two-stage calibration of a 6-axis force-torque sensor for robust operation in the mars 2020 robot arm. *Advanced Robotics*, 35(21-22):1347–1358, 2021.

[23] Dieter Fox Sebastian Thrun, Wolfram Burgard. *Probabilistic Robotics*. Massachusetts Institute of Technology, 2006.

[24] Yufeng Sun, Lin Zhang, and Ou Ma. Toward a framework for levels of robot autonomy in human-robot interaction. *Journal of human-robot interaction*, 3(2):74–99, 2014.

[25] Yufeng Sun, Lin Zhang, and Ou Ma. Force-vision sensor fusion improves learning-based approach for self-closing door pulling. *IEEE Access*, 9:137188–137197, 2021.

[26] Richard Szeliski. *Computer Vision: Algorithms and Applications 2nd Edition*. Springer Cham, September 2021.

[27] Kenneth Waldron and James Schmiedeler. *Kinematics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[28] David P. Watson and David H. Scheidt. Autonomous systems. *Johns Hopkins APL Technical Digest*, 26(4):368–376, 2005.

[29] Yizhou Zhao, Qiaozi Gao, Liang Qiu, Govind Thattai, and Gaurav S. Sukhatme. Opend: A benchmark for language-driven door and drawer opening, 2022.

Appendix A

A.1 World Model Ontology

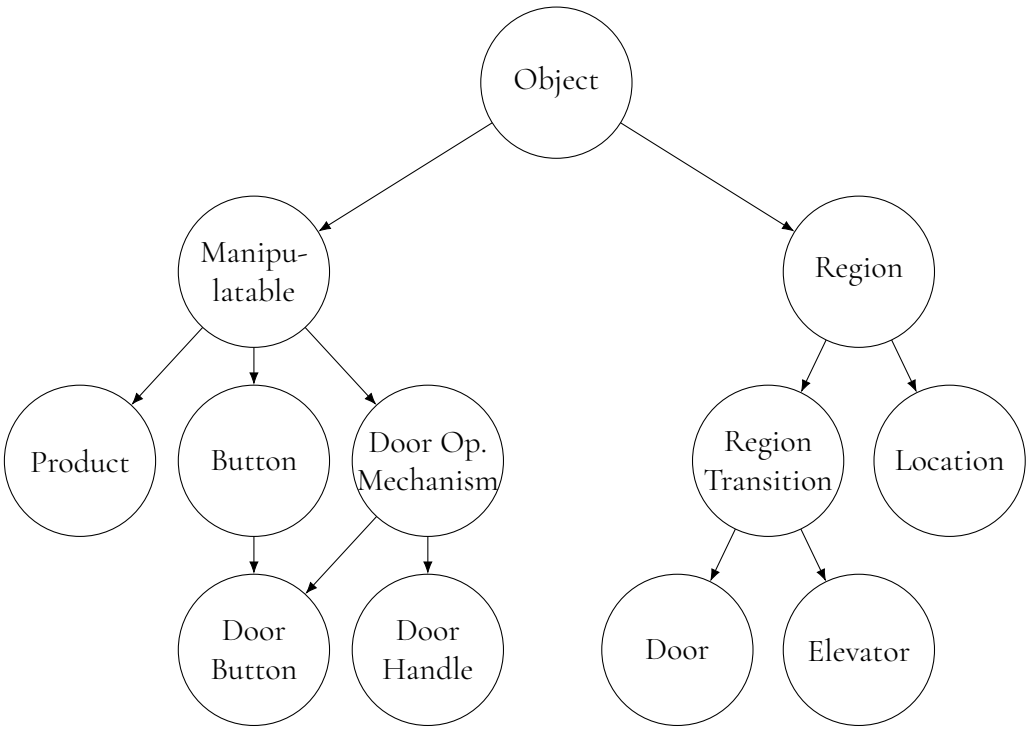

Figure A.1: The ontology of object types.

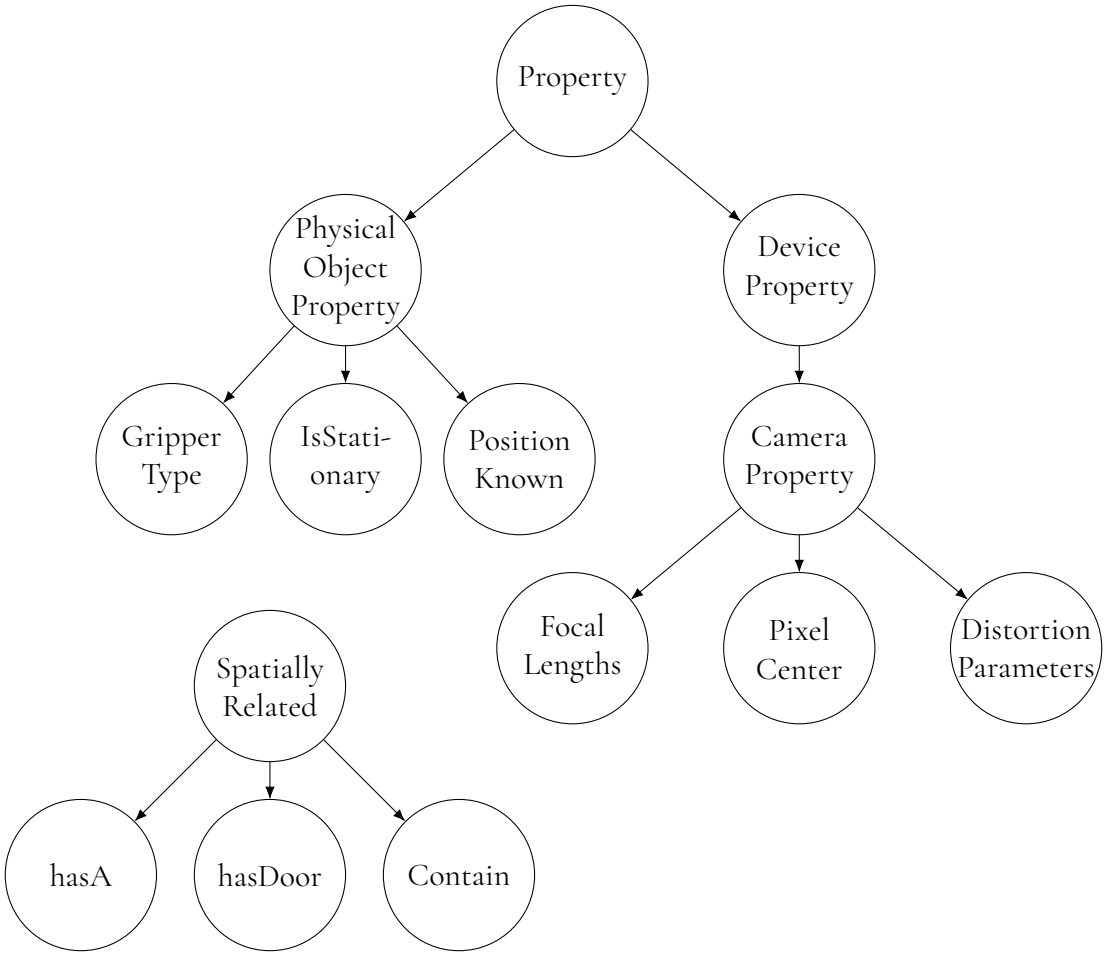

Figure A.2: The ontology of object properties and object relations.

A.2 Demo Video

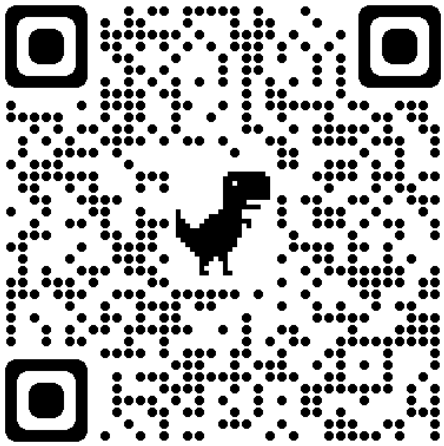

Figure A.3: QR code containing a link to a demo video which showcases the door opening and door passing skills that were implemented for Heron.

Figure A.4: QR code containing a link to a demo video which showcases the button pressing generalizing to different buttons.

A.3 Code

The code can be found here: https://github.com/p-rosit/AutonomousDoorOpening

EXAMENSARBETE Skill-Based Autonomous Door Opening
STUDENT Josefin Gustafsson, Pontus Rosqvist
HANDLEDARE Volker Krüger (LTH)
EXAMINATOR Jacek Malec (LTH)

Dörröppning - Hur en robot kan uppnå en treårings förmåga

POPULÄRVETENSKAPLIG SAMMANFATTNING **Josefin Gustafsson, Pontus Rosqvist**

Detta projekt utvecklar en färdighet för en autonom robot. Färdigheten tillåter en robot att hitta positionen av en dörrknapp, öppna dörren och passera denna. Projektet är utfört med mjukvaran ROS och SkiROS.

Robotar i dagens samhälle används mest i industrin där de är separerade från människor. Som tekniken utvecklar sig i dagens samhälle gör robotarna en förflyttning till platser som är designade för människor. För att kunna existera i samma utrymme måste de kunna navigera i dessa områden. Detta ger ett behov av en navigeringsfärdighet som ger en robot möjligheten att inte bara ta sig runt i rum utan även mellan dessa. På grund av detta har vi implementerat en färdighet för detta.

Färdigheter är byggnadsblock som en robot kan använda för att utföra olika processer. Med hjälp av SkiROS kan vi fokusera på att endast programmera roboten utan att tänka på hur informationen runtomkring ska hanteras. En färdighet är inte specifik till en viss robot eller en viss situation utan den ska beskrivas på ett sätt som kan användas i ett stort antal situationer. En grafrepresentation av byggnaden har konstruerats som beskriver hur de olika rummen relaterar till varandra. Dörrar och deras knappar finns även i denna grafen. Grafen kan sedan användas för att, givet en start och ett mål, planera vilka dörrar roboten kommer att behöva passera.

Roboten använder den planerade vägen och följer denna. Vid passering av en dörr måste en knapp hittas vilket sker med en kamera på armen som hittar positionen där det ska tryckas. Armen rör sig sedan till knappen och trycker på den. Att knappen är tryckt avgörs med hjälp av kraftsensorer. För att försäkra att dörren faktiskt har öppnats har en färdighet som detekterar om dörren är öppen eller stängd gjorts. Efter detta följer roboten återigen navigeringsplanen och går igenom dörren. Dessa beskrivna färdigheter kan sättas ihop för att navigera i en byggnad. Den totala färdigheten sätter sedan ihop alla dessa byggnadsblock till en fullständig navigeringsfärdighet.

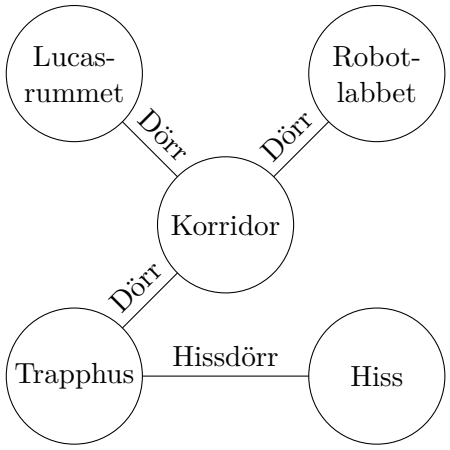

Figure 1: Grafrepresentation av byggnaden