Institut für Parallele und Verteilte Systeme

Abteilung Maschinelles Lernen und Robotik

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Master Thesis Nr. 66

Active Exploration and Identification of Kinematic Devices

Jochen Mohrmann

Studiengang:	Informatik
Prüfer:	Prof. Dr. rer. nat. Marc Toussaint
Betreuer:	Prof. Dr. rer. nat. Marc Toussaint
begonnen am:	11.11.2015
beendet am:	11.05.2016
CR-Klassifikation:	I.2.6, I.2.9, I.2.11, G.1.6

Abstract

As an important part of solving the lockbox problem, this thesis deals with the problem of identifying kinematic devices based on data generated using an Active Learning strategy. We model the belief over different device types and parameters using a discrete multinomial distribution. We discretize directions as a Geodesic sphere. This allows an isotropic distribution without being biased towards certain directions. The belief update is based on experience using a Bayes Filter. This allows to localize the correct states, even if an action fails to generate movement. Our action selection strategy aims to minimize the number of actions necessary to identify devices by considering the expected future belief. We evaluate the effectiveness of different information measures and compare them with a random strategy within a simulation. Our experiments show that the use of the *MaxCE* strategy creates the best results. We were able to correctly identify *prismatic*, *revolute*, and *fixed* devices in 3D space.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

1 Introduction

1.1 Motivation

Problem solving is the process of finding solutions to issues of a certain complexity [Oxf16]. It is a form of intelligent acting. Its aim is to transform a given state into a goal state. Not only for human beings this involves a cognitive process, animals possess these capabilities as well. Auersperg, Kacelnik, and von Bayern [AKB13] showed this by demonstrating that Goffin's cockatoos (*Cacatua goffini*) are able to solve means-means-end problems. These problems require a sequence of actions (means) in order to reach the desired goal state (end) [SRS+05]. The birds were confronted with a box that contains a reward in form of food. The box is locked by five inter-locking devices. In order to open it, they have to be unlocked in the correct order. Figure 1.1 shows a lockbox of increased size.

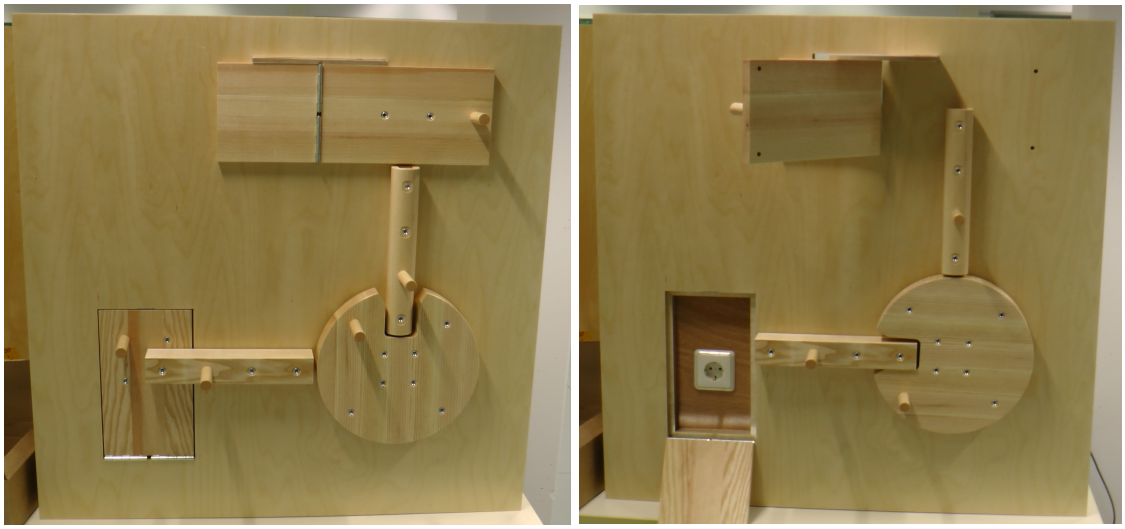

Figure 1.1: The lockbox with increased size and different locks to fit the needs for the PR2 robot. The left picture shows the box in a closed state, whereas on the right the box is opened and reveals a power outlet.

Although some of them are physically similar in nature, it is necessary to operate them differently. The first lock is a simple pin. It has to be pulled out of a hole to enable the movement of the next lock. This second lock, a screw, has to be rotated in order to obtain a translational movement. The third lock, a bolt, has to be pushed through a fixation ring. The fourth lock is a wheel. It has to be rotated around its axis to the correct position, before it can be entirely removed. A bar is the last device. In order to open the door, it has to be pushed in the correct direction. Each lock can be operated by the bird using beak or claw.

The experiment was conducted with different settings. One subject was able to solve the problem with all five locks present and without previous knowledge about individual locks or dependencies. Others were able to imitate this after social demonstration. Interestingly, once a bird managed to solve a specific setting, it seldom failed on subsequent trials. Additionally, they were able to react flexibly towards alteration of the settings.

The birds proof that they have a variety of skills:

1. Knowing how different locks have to be operated in order to open them
2. The necessary movement skills to actually open a lock
3. Understanding interactions between locks
4. Transferring knowledge between different settings

The goal is to enable a PR2 robot, as seen in Figure 1.2, to open the lockbox. This can be achieved using different approaches.

For example, it is quite possible to open the box using teleoperation. A human operator can perform movements that are mimicked by the robot. He can dictate the robot's movements to open the locks in the correct order. Knowledge about the lockbox however resides only on the human operator. A naive operator may have to learn the locks and dependencies similar to the birds, the robot on the other hand is only used as a tool.

Another approach is to guide the robot to open the box by demonstrating it to him. A teacher can guide the robot by manipulating its joints to perform movements that lead to the opening of locks and finally to solving the lockbox. In fact, the manipulation can be executed using teleoperation. With this supervised learning technique, a robot can learn the necessary movement skills to operate the lockbox given a specific setting. Additionally, the robot can optimize the trajectories towards particular goals, like minimizing joint accelerations. Applied to a lower level, the robot can learn to operate single locks, too. However, this technique still doesn't allow the robot to actually understand lock interaction. In order to go from the ability to open locks towards learning dependencies, another step is necessary. This method is similar to the social learning the birds

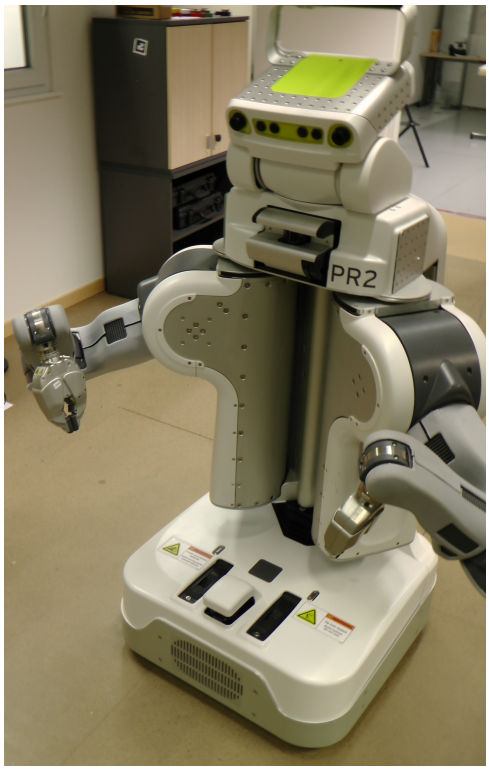

Figure 1.2: The PR2 robot.

apply. Instead of learning everything from scratch, a supervisor provides the necessary knowledge.

There was however a subject that was able to accomplish the goal without external help or prior knowledge. To develop this behavior, the robot has to actively explore the whole environment. The skill to reason about dependencies and how actions influence the environment are important as well. To solve this problem, it can be subdivided into multiple smaller problems that build on each other.

1. On the highest level, the parameters of individual locks are well known. This includes the knowledge of key points in the joint configuration space which lock or unlock a device. The level deals with the dependencies between different devices. Which requirements have to be fulfilled to disengage a specific lock and ultimately open the lockbox's door? This level can be described as a symbolic problem.

2. Given the general parameter of a device, the goal for this level is to determine under which circumstances it can be described as open or closed. Here a device can be considered individually.

3. On this level, the general parameters of a device have to be learned. This includes its location, joints and possible limits. The nature of the lockbox problem however

doesn't allow to simply learn parameters of individual locks and then assume they are fixed. Since a lock may only be unlocked later on in the process, its general parameters, especially its limits, may change depending on other devices. Therefore, it may be necessary to constantly reevaluate and update the parameters and describe them as depending on other locks.

4. Before a device can be operated and learned, it has to be identified. Given pictures, videos or other sensor data, potential spots of interest have to be found. This problem is part of the area of Computer Vision.

5. A problem that occurs throughout nearly all levels is the question of how to actually manipulate an object. This motion problem is of relevance when learning how individual devices work, but is also necessary for the implementation of the symbolic action of opening a specific lock within the first level.

Actual solutions for the problems may approach multiple levels at once. For example, it is inconvenient to find out how to lock or unlock a device, if other locks are not taken into account. Kulick, Otte, and Toussaint [KOT15a] assume that locks give feedback to signal key point in the joint configuration space. In the end however, one can only be certain of the status, if the locked object is now actually free. Additionally, a device may function in such a way that it can lock multiple other devices or only a subset depending on its configuration. In this case a binary decision between locked and unlocked is only possible towards other devices, not in general. Hence, it may be necessary to solve the first two levels combined.

1.2 Limitations and Future Goals

An important demand for robots is to model and learn the environment they operate in. One way is describing it as a set of rigid parts linked by joints. Afterwards, a robot is able to learn to manipulate the environment's degrees of freedom [KOT15a]. This involves simpler tasks like accessing a drawer by pulling or more sophisticated actions like opening a door by turning a key, pulling the handle, and pushing it open. The latter problem further requires modeling of dependencies between joints. Certain joints may only be accessible, if others are set correctly. In a way, this is exactly the same as understanding how locks work and how they inter-lock each other. Ultimately enabling a robot to solve the lockbox by understanding its degrees of freedom can be transferred to other real world problems, if it is possible to generalize the essential skills.

There is already previous work regarding the lockbox problem. On a higher level Kulick, Otte, and Toussaint [KOT15b] show that the problem can be solved using *Monte-Carlo tree search (MCTS)* in the belief space. They assume that the individual joints are already

well known. The goal is to discover the dependency between joints. This structure is modeled as a probability distribution based on the work of Kulick, Otte, and Toussaint [KOT15a]. Actions correspond to moving the mechanism to a position. Instead of considering all possible actions, the MCTS algorithm considers only three actions per joint. One that most likely locks the joint, one that most likely opens it and the one with the maximum expected cross entropy over the belief before and after the action is taken. Using the cross entropy as intrinsic reward alongside the reward for opening the lockbox, this approach shows promising results. However, it relies on the knowledge about all joints and the corresponding parameters.

The problem of finding and learning this joints is called *kinematic identification*. This is the purpose of this work, to identify certain types of joints and learn their parameters. We assume that information about the environment is available in form of translations and rotations for a given object. Additionally, we assume that points of interest are already identified and we only need to learn, whether it is actually a joint or rigid. In Chapter 2 we present previous work done in this area of research. Most of the work is focused on identifying joints based on given movement. Without previous knowledge, manipulating the objects in the first place to generate the required movement, is a complex problem. Barragán, Kaelbling, and Lozano-Pérez [BKL14] solve this problem by introducing an effective Active Learning strategy. However, they consider only 2D devices.

By combining these approaches, the robot will hopefully be able to autonomously open the lockbox, without previous knowledge. Then, we are able to generalize our method to let the robot learn new environments of all sorts.

1.3 Contributions

This work implements the *kinematic identification* method as described by Barragán, Kaelbling, and Lozano-Pérez [BKL14] in order to identify the types, parameters, and variables of locks without previous motion. We enhance this work by the following contributions:

1. Implementation for the 3D space, instead of a 2D setting.
2. Introduction of a new discretization strategy based on the geodesic sphere.
3. Introducing the cross-entropy as new measure for information gain which doesn't support wrongly biased beliefs.
4. Building simple transition models for the *prismatic* and *revolute* devices.

1.4 Structure

Chapter 2 is an overview of the work done in the area of *kinematic identification*. The approaches differ in how they model the problem and how the necessary data is generated, by demonstration, given trajectories or autonomously letting the robot select actions. In section 3, we introduce the background, which may be necessary in order to understand this work. Chapter 4 describes our problem in a mathematical way, the actual implementation is given in section 5. In Chapter 6, we present the results of conducted experiments to evaluate our approach. Finally in 7, we discuss the results of our work, present alternative approaches and insights, and draw a conclusion.

2 Related Work

The lockbox problem itself is already approached on a higher level by [KOT15b]. In order to actually apply the proposed strategies, the kinematic devices locking the box have to be known and understood in a way that allows for exact manipulations. One part is therefore to identify the devices and distinguish them based on their functionality. This problem is known under the term *kinematic identification* and previous work reveals multiple strategies to tackle the problem.

Katz and Brock [KB08] use a vision-based approach and build feature clusters with respect to their relative motion. Based on this, the underlying joint type, revolute or prismatic, can be identified, connecting the respective parts of the mechanism. While this approach uses the robot and performs predefined motions, Pillai, Walter, and Teller [PWT15] use human demonstration to generate movement.

Sturm, Stachniss, and Burgard [SSB11] present a framework for the creation of a probabilistic, kinematic model of articulated objects. Joints are modeled as rigid, prismatic, revolute or Gaussian process and in combination form a kinematic graph. The data-driven Gaussian process model allows to identify joints that can not explicitly be modeled by the other three types. They are identified from object trajectories. Using the Bayesian Information Criterion we can identify the best model for each joint without favoring more complex types. Given the model its parameters are calculated using the maximum mean-estimator.

Martín and Brock [MB14] use a Bayes Filter to perform model estimation online and reason about uncertainty. In contrast to [SSB11], it detects moving rigid bodies and does not require to know them beforehand. Using three interconnected levels of recursive estimation, the problem is simplified into easier to solve subproblems. At the lowest level, sensor data are used as measurement to detect feature movement. This in turn allows to estimate rigid body motion and finally the kinematic state. In this holistic approach, feedback is given to the lower levels in form of state predictions. All these approaches have in common that they rely on visual data and require existing motion. Their success depends largely on the quality of this information. If certain joints are not manipulated, they can not be identified.

An alternative is to use the end-effector position of a robot's manipulating the object. Sturm et al. [SJS+10] use the equilibrium point control (EPC) to command the robots

manipulator to a certain point. This point is the Cartesian-space equilibrium point (CEP). In case of absence of externally applied forces, other than gravity, the robot's manipulator will end up at the CEP. Keeping the handle of the considered object attached to the manipulator, the position difference between the CEP and the actual position can be used to update the probabilities over different joint types. This process is repeated for a trajectory of CEPs. While this approach is a form of action selection, it still relies on the initial trajectory. There is no need for the agent to plan its actions.

Katz, Pyuro, and Brock [KPB09] model the world as a relational state representation. Using the three predicates $R(\cdot)$, $P(\cdot)$ and $D(\cdot)$, for a revolute, prismatic, or disconnected joints, the relation between rigid bodies can be described. Relational reinforcement learning, in form of Q-Learning, is used to find policies that map actions to such relational state descriptions. As a reward, the number of newly discovered joints has been used. For the actual identification of joint relations, a vision based approach is used [KB08]. Their results show that this learning-based action selection strategy significantly reduces the number of actions necessary to correctly identify kinematic devices.

Narayanan and Likhachev [NL15] used the kinematic graph introduced by [SSB11]. They extend the graph to allow joint types being represented as a function of their pose, which allows to model lock-like devices. For example, depending on the pose of a door handle, the joint between a door and a frame can be described as rigid or revolute. In contrast to other works, they explicitly formulate a goal state and try to reach it while simultaneously identifying the device. The proposed planning algorithm assumes a deterministic transition model for a known mechanism and a completely observable object state and allows for real-time execution. However, they initialize their belief based on a visual system that limits the number of possible mechanism types.

Barragán, Kaelbling, and Lozano-Pérez [BKL14] use a Bayes Filter to identify kinematic devices. In contrast to other approaches, it does not rely on visual information in order to identify joints. Information like force feedback or sounds can be used alternatively or in addition. The type of a device is determined without directly identifying individual joints using a black box approach. Additionally, they incorporate serial mechanisms, which allows for different behaviors under different configuration states similar to [NL15]. The authors also introduce an Active Learning technique that allows to calculate actions that are expected to maximize information in form of entropy. As opposed to previously described techniques, the action selection does not only depend on the mean estimator over the devices, but actually considers the whole belief range. This selection allows to find significant actions, even if there is no initial information available. This work uses the proposed strategy from Barragán, Kaelbling, and Lozano-Pérez [BKL14] and applies the same Bayes Filter. It extends the possible configurations of the kinematic devices to the 3D space and introduces a new information measure from Kulick, Lieck, and Toussaint [KLT14].

Hausman et al. [HNOS15] use the work of Sturm, Stachniss, and Burgard [SSB11] and combine it with a Bayesian Filter similar to Barragán, Kaelbling, and Lozano-Pérez [BKL14]. They use the same Active Learning strategy, however, they do not represent the belief by discretizing the variables and parameters, but by introducing a particle filter, which allows for state spaces with higher dimensionality.

3 Background

3.1 Notations

We will refer to variables with upper case letters, like X, and to its realization using lower case letters, like x. The probability $P(X = x)$ of X taking on the value x is shortened to $P(x)$ for simplicity.

3.2 Hidden Markov Model

A hidden Markov Model (HMM) [CB03] models a system, which is assumed to be a Markov process with unobserved (hidden) states. The states are represented by a set of random variables $Q_{1:T} = \{Q_1, \ldots, Q_T\}$, with realizations from the state space $q \in S$. Only the realization of the set of random variables $Y_{1:T} = \{Y_1, \ldots, Y_T\}$ is observable, we refer the these variables as observations. Let O be the space of possible observations. For the states the Markov property holds true, which means that the conditional probability distribution of future states of the process only depend upon the current state and are independent of the current time t:

$$P(q_t|q_{1:t-1}) = P(q_t|q_{t-1}). \tag{3.1}$$

The observations are only dependent on the current state

$$P(y_t|q_{1:t}, y_{1:t-1}) = P(q_t|q_{t-1}) \tag{3.2}$$

and are independent towards other observations:

$$P(y_{t:t+h}|q_{t:t+h}) = \prod_{i=t}^{t+h} P(y_i|q_i). \tag{3.3}$$

We define

(3.4) $\pi_i = P(Q_1 = i)$

to be our initial state distribution, where $i \in S$. The transition model gives the probabilities for changing states:

(3.5) $P(Q_t = i|Q_{t-1} = j),$

where $i, j \in S$. The observation model gives the probability of an observation, given the current state:

(3.6) $P(Y_t = i|Q_t = j),$

where $i \in O$ and $j \in S$.

We can extend the HMM model to an Input-Output Hidden Markov Model (IOHMM), by conditioning our states and observations on input variables $X_{1:T} = \{X_1, \ldots, X_T\}$:

(3.7) $P(q_t|q_{1:t-1}, x_{1:t}) = P(q_t|q_{t-1}, x_t),$

and

(3.8) $P(y_t|q_{1:t}, y_{1:t-1}, x_{1:t}) = P(q_t|q_{t-1}, x_t).$

The initial state distribution is changed accordingly to

(3.9) $\pi_i = P(Q_1 = i|x_1),$

the transition model to

(3.10) $P(Q_t = i|Q_{t-1} = j, x_t),$

and the observation model to

(3.11) $P(Y_t = i|Q_t = j, x_t).$

The general form of the HMM assumes a discrete state space S. We omit this generalization, because we want to allow continuous variables in S.

3.3 Quaternion

Let Q be a *quaternion* [Vin11] of the form

(3.12) $Q = w + ix + jy + kz,$

where the *constituents* $w, x, y, z \in \mathbb{R}$ and i, j, k are *imaginary units* for which the following equations hold true:

(3.13) $i^2 = j^2 = k^2 = ijk = -1$

and

(3.14) $ij = k \qquad jk = i \qquad ki = j \qquad ji = -k \qquad kj = -i \qquad ik = -j$

We can rewrite the quaternion Q as an ordered pair, a combination of a scalar w and vector part $\vec{v}$,

(3.15) $Q = (w, \vec{v}),$

where $w \in \mathbb{R}$ and $\vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3$.

There are three operations on the set of quaternions. The sum of two quaternions is the sum of their elements in $\mathbb{R}^4$.

$$Q_1 + Q_2 = (w_1, \vec{v}_1) + (w_2, \vec{v}_2) = (w_1 + w_2, \vec{v}_1 + \vec{v}_2) \tag{3.16}$$

The multiplication of two quaternions is given by:

$$Q_1 Q_2 = (w_1, \vec{v}_1)(w_2, \vec{v}_2) = (w_1 w_2 - \vec{v}_1 \cdot \vec{v}_2, w_1 \vec{v}_2 + w_2 \vec{v}_1 + \vec{v}_1 \times \vec{v}_1), \tag{3.17}$$

where $\cdot$ is the dot product and $\times$ the cross product. The multiplication is associative, but not commutative.

The scalar multiplication can be interpreted as a quaternion multiplication, where the scalar factor λ corresponds to a *real* quaternion. A *real* quaternion has a vector part $\vec{0}$.

$$\lambda Q = (\lambda, \vec{0})(w, \vec{v}) = (\lambda w, \lambda \vec{v}) \tag{3.18}$$

The inverse Q^{-1}, norm $|Q|$ and conjugate Q^* of Q are given by:

$$Q^* = (w, \vec{v})^* = (w, -\vec{v}) \tag{3.19}$$

$$|Q| = |(w, \vec{v})| = \sqrt{w^2 + \vec{v}^2} \tag{3.20}$$

$$Q^{-1} = \frac{Q^*}{|Q|^2} \tag{3.21}$$

We can use quaternions as an efficient way to rotate a point p in 3D-space. We represent p as a *pure* quaternion P. A *pure* quaternion has a scalar part which is equal to zero.

$$P = (0, p) \tag{3.22}$$

We construct Q to be a rotation quaternion that allows to rotate p by an angle θ around an arbitrary axis $\hat{v}$:

$$Q = (\cos\frac{1}{2}\theta, \sin\frac{1}{2}\theta\hat{v}) \tag{3.23}$$

The rotated point p' is obtained by multiplying Q with P and the inverse Q^{-1}:

$$P' = (0, p') = QPQ^{-1} \tag{3.24}$$

Another operator we use in this thesis is the inner product of two quaternions:

$$Q_1 \cdot Q_2 = (w_1, \vec{v}_1) \cdot (w_2, \vec{v}_2) = w_1 w_2 + \vec{v}_1 \cdot \vec{v}_2 \in \mathbb{R} \tag{3.25}$$

4 Problem Formulation

The objective it to identify the correct *type* and *parameter* of a kinematic device. Section 4.1 describes how a probabilistic model over the parameter and variable space of the considered device types are build. Actions are chosen using Active Learning techniques in Section 4.3. The resulting device behavior is then used to update the probabilistic model as described in section 4.2.

4.1 Probabilistic Model

Barragán, Kaelbling, and Lozano-Pérez [BKL14] formulate the problem as a discrete-state, Input-Output Hidden Markov Model (IOHMM). The hidden states are the possible states of a kinematic device. Inputs are *actions* performed on the device and outputs are sensed *observations*.

A *state* is a triple $s = (m, \theta, x) \in S$.

- m is the *type* of the device. In principal, those *types* can be associated with arbitrary classes.
- θ is a vector of *parameters*. The number and meaning of parameters depend on the considered *type*. It characterizes the properties of a device.
- x is a vector of *variables*. Similar to θ it depends on m. It describes the current configuration of the device.

An *action* can be anything the robot can do in order to get information about a device. They have to be executable independently of the device *state* in general and in particular its *type* and within finite time. It is assumed that *actions* always terminate and run to completion. *Actions* may not only be limited to actually manipulating the device by pushing or pulling, but can include looking from different angles.

Observations can be different measures of the environment through sensors. This may include vision, tactile, force, position, vibration, and sound sensors.

Under the assumption of a discrete-time model, the *transition model* models the relationship between *state* and *action* as a conditional probability distribution over *state* values at time $t+1$, given the *state* and *action* at time t.

$$
\begin{aligned}
P(s_{t+1}|s_t, a_t) &= P(m_{t+1}, \theta_{t+1}, x_{t+1}|m_t, \theta_t, x_t, a_t) \\
&= P(m_{t+1}, \theta_{t+1}|m_t, \theta_t, x_t, a_t) \cdot P(x_{t+1}|m_{t+1}, \theta_{t+1}t, x_t, a_t)
\end{aligned} \tag{4.1}
$$

The model describes two different behaviors. *Type* and *parameter* changes are modeled using Eq. (4.2).

$$P(m_{t+1}, \theta_{t+1}|m_t, \theta_t, x_t, a_t) \tag{4.2}$$

Since this model assumes changes of the underlying characteristics of a device, the likelihood for an actual change should be very small. In this work, we assume that such behavior does not occur. It would imply damaging the device or altering it in a way, which is not within our scope. Therefore, we assume the probability of changing the *type* or *parameter* is zero. For exploring inter-lock dependencies however, this behavior is common and desired, since its goal is to remove restrictions imposed from other locks.

Eq. (4.3) models the expected behavior of a device. It describes the change of *variables* of a device of *type* m and *parameters* θ after executing *action* a.

$$P(x_{t+1}|m_{t+1}, \theta_{t+1}, x_t, a_t) \tag{4.3}$$

The observation model is given in Eq. (4.4). It determines the likelihood for an observation vector, given the current state and previous action.

$$P(o_{t+1}|m_{t+1}, \theta_{t+1}, x_{t+1}, a_t) \tag{4.4}$$

4.2 Belief Update

Given a *posterior* belief in the state space, $P(s_0)$, an action sequence, $a_{0:T-1} = a_0, \ldots, a_{T-1}$, and the resulting observations, $o_{1:T} = o_1, \ldots, o_T$, the objective is to calculate the *posterior* belief over *type* and *parameter* of the device at time T, $P(m_T, \theta_T | a_{0:T-1}, o_{1:T})$. Eq. (4.5) can be obtained by marginalizing out the variables at time T and marginalizing over the states from all time steps:

$$\begin{aligned} P(m_T, \theta_T | a_{0:T-1}, o_{1:T}) &= \sum_{x_T} P(m_T, \theta_T, x_T | a_{0:T-1}, o_{1:T}) \\ &= \sum_{x_T} P(s_T | a_{0:T-1}, o_{1:T}) \\ &= \sum_{x_T} \sum_{s_{0:T-1}} P(s_{0:T} | a_{0:T-1}, o_{1:T}) \end{aligned} \tag{4.5}$$

Using the conditional independence relationship from the IOHMM, one can factor out the individual probabilities for the state changes at different time steps:

$$P(m_T, \theta_T | a_{0:T-1}, o_{1:T}) = \sum_{x_T} \sum_{s_{0:T-1}} P(s_0) \prod_{t=0}^{T-1} P(s_{t+1} | s_t, a_t, o_{t+1}) \tag{4.6}$$

The observation model only describes the probability of making an observation given the state and not vice versa. Therefore, Bayes' rule is applied:

$$P(m_T, \theta_T | a_{0:T-1}, o_{1:T}) \propto \sum_{x_T} \sum_{s_{0:T-1}} P(s_0) \prod_{t=0}^{T-1} P(o_{t+1} | s_{t+1}, a_t) \cdot P(s_{t+1} | s_t, a_t) \tag{4.7}$$

For the one-step update with $T = 1$, the equation can be simplified:

$$P(m_1, \theta_1 | a_0, o_1) \propto \sum_{x_1} \sum_{s_0} P(s_0) \cdot P(o_1 | s_1, a_0) \cdot P(s_1 | s_0, a_0) \tag{4.8}$$

4.3 Active Learning

There is a variety of different strategies that deal with the problem of selecting appropriate actions. In order to compare them, we need to define a notion of a desired outcome in terms of the probability distribution.

We can interpret our current belief as a random experiment, where our state space corresponds to a random variable S. Drawing a sample s from the belief results in a device with given *type*, *parameters* and *variables*. This corresponds to an event. The information of the event s can be defined to the base d as:

$$i(s) = -\log_d P(s) \tag{4.9}$$

The entropy is the expected information and is a measure of disorder. In the discrete case it is defined as [Sha01]:

$$H(S) = E[i(S)] = \sum_s P(s)i(s) = -\sum_s P(s)\log_d P(s) \tag{4.10}$$

Barragán, Kaelbling, and Lozano-Pérez [BKL14] propose the conditional entropy over *type* and *parameters*, $H(m_T, \theta_T | a_{0:T-1}, o_{1:T})$, as a measure of how well a device is *known*.

While a random action selection is easy to implement and understand, it has the drawback of not aiming to minimize this measure. The number of actions necessary in order to reduce the conditional entropy below a certain threshold may be very large.

Planning actions to reduce the entropy is beneficial. The optimal strategy is to minimize the expected conditional entropy after the whole sequence of actions. This, however, is a very complex problem. Instead, we focus on one-step heuristics that only takes the next action into account.

Barragán, Kaelbling, and Lozano-Pérez [BKL14] use such a one-step strategy. Consider a possible action a at time T. Based on the current belief b, the *transition model*, and the *observation model*, it is possible to calculate the expected belief b' over all possible *observations* o after executing this action. The action a^* which leads to the belief with the lowest entropy is chosen:

$$a_T^* = \operatorname*{argmin}_a E_{o|b,a} H(s_{T+1} | a_{0:T}, o_{1:T}, a, o) \tag{4.11}$$

In Section 5.5.1 we describe a problem that may emerge depending on the implementation of the discretization. In such cases, the attempt to minimize the entropy myopic, may cause undesired effects. They manifest in selecting actions that aim to preserve the current belief, rather than gaining additional information. Therefore, an additional one-step criterion is introduced: the *maximum cross-entropy* (*MaxCE*) [KLT14]:

$$
\begin{aligned}
a_T^* &= \underset{a}{\operatorname{argmax}}\, E_{o|b,a} H[P(s_T|a_{0:T}, o_{1:T}); P(s_{T+1}|a_{0:T}, o_{1:T}, a, o)] \\
&= \underset{a}{\operatorname{argmax}}\, E_{o|b,a} D_{KL}(P(s_T|a_{0:T}, o_{1:T}) \;||\; P(s_{T+1}|a_{0:T}, o_{1:T}, a, o))
\end{aligned} \tag{4.12}
$$

$H[P(Z), Q(Z)]$ is the Kullback-Leibler divergence (KLD) and quantifies the additional information captured in the posterior $P(s_{T+1}|a_{0:T}, o_{1:T}, a, o)$ to the prior $P(s_T|a_{0:T}, o_{1:T})$. Again, the equation is given for the discrete case:

$$
H[P(Z), Q(Z)] = -\sum_z P(z) \log Q(z) \tag{4.13}
$$

This criterion allows to select actions leading to higher entropy in the expected posterior. Even though this may seem counter-intuitive, we will show in Section 6.2, that this criterion is superior and outperforms the plain entropy in our scenario.

5 Implementation

The lockbox problem includes different kinds of kinematic devices. If inter-locking is ignored, each device can be characterized as one of five movement constraining mechanisms. They can be modeled and distinguished by device parameters and variables. Parameters describe the lock and are independent from its current status. In this work, we assume that parameters are constant. Variables describe the current state of the lock.

1. The **free** model enforces no constraints and therefore has no parameters. Its variables are the position and pose in the 3D space.
2. The **fixed** or **static** model allows no motion. It is rigid towards the world frame and its parameters are location and pose. There are no variables.
3. The **prismatic** model allows 1-DOF motion along a given axis. The movement is limited to certain range along this axis. Parameters of the model are the 3D coordinates of one of the two limits and the axis. An additional parameter is the maximum displacement from the first limit along the axis. This indirectly gives the coordinates of the second limit. The only variable is the current displacement. The prismatic model does not allow for any kind of rotation.
4. The **revolute** model allows 1-DOF rotation around a given axis. Parameters are the coordinates of the center, the rotation axis, and the radius. The only variable is the rotation angle.
5. The **screw** model is a combination of the prismatic and revolute model. By rotating the disk around the axis, it is translated along this axis. Parameters are the rotation axis, 3D coordinates of both limits, and a constant that describes the relation towards rotation and translation. The only variable is the current prismatic displacement.

This thesis discusses a method for distinguishing between three kinds of devices: The prismatic, static, and revolute models. Table 5.2 summarizes variables and parameters for these models. However, the presented methods can be applied to additional types accordingly.

model	parameter θ	variables x
prismatic	$x_{limit}, y_{limit}, z_{limit}, x_{axis}, y_{axis}, z_{axis}, d_{max}$	d (displacement)
static	$x_{pos}, y_{pos}, z_{pos}$	-
revolute	$x_{center}, y_{center}, z_{center}, x_{axis}, y_{axis}, z_{axis}, r$	α (rotation angle)

Table 5.1: State space of kinematic devices

5.1 Actions and Observation Model

In our context, *actions* are a constant *force vector* v that is constantly applied to a specific *position* p on the device for a short time period. This corresponds to the robot using its end effector to push the object. In order to apply the constant force, a PD-controller can be used. The time period is assumed to be short enough that the robot can maintain the force direction.

Observations are the translation in form of a vector and the rotation given as a quaternion applied to the considered point after the action is executed. When executed on the robot, the *observations* will come from a vision module that additionally provides uncertainty of the measurements in form of standard deviation. For the simulations we perform in this work, the *transition model* is used for estimating device behaviors is also used to estimate observation.

5.2 Discretization of the State Space

A device is represented by a model for each device *type*. For each *type* a different discretized state space is used, which in return depends on its *parameters* and *variables*. The belief is therefore a multinomial distribution. Since the prismatic and revolute model both share the axis as a common property, it is described separately. The goal of this section is to describe the considered state space for each model *type* m. A state can be described using m and a sub-state that depends on m: $s = (m, s_m) \in S$. The belief is initialized uniformly over m, s.t. the summed probabilities over sub-states are equal.

5.2.1 Axis

The axis is only an indicator for a direction in 3D-space. Modeling it in the same space is therefore unnecessary. Instead, the belief can be build and discretized over a sphere. The sphere is modeled using a geodesic grid [WKO92]. This is a more complex strategy

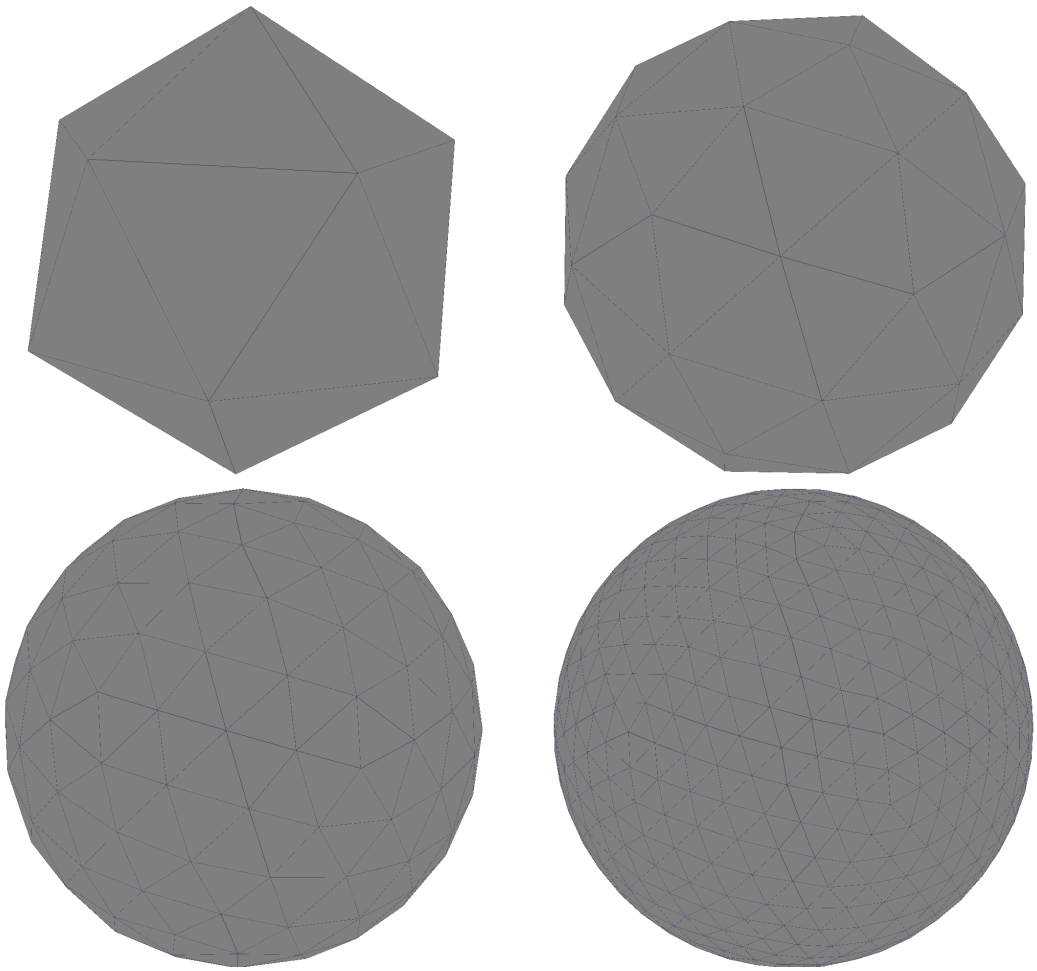

Figure 5.1: The pictures show the Geodesic sphere with different resolutions. From left to right: Icosahedron, Geodesic sphere after one two and after three subdivisions.

than sampling rectangular grids along the latitude and longitude like it is commonly done for the Earth's surface. But since there are no preferred directions, the points of the discretized sphere should be distributed isotropically. Additionally, it is simple to increase the resolution of the discretization using binary division.

The 12 vertices of the convex regular icosahedron, which is circumscribed by the unit sphere around the origin, are calculated. This way the vertices have a length of one and can be considered as normalized direction vectors. The 20 triangular faces are obtained by combining three neighboring vertices. In order to increase the resolution, subdivisions can be performed, where each face is divided into four smaller triangles. The new vertices are projected onto the unit sphere. The process is described in Algorithm 5.1. Figure 5.1 shows the results.

Algorithmus 5.1 SubDivide Algorithm

```
1: function SUBDIVIDE(V,T)      // V is a set of vertices, T is a set of triangular faces
2:     V* ← V
3:     T* ← ∅
4:     for all t ∈ T do
5:         (a, b, c) ← getVerticesFromTriangle(t, T)
6:         d ← (a+b)/2
7:         e ← (a+c)/2
8:         f ← (b+c)/2
9:         d ← projectOntoUnitSphere(d)
10:        e ← projectOntoUnitSphere(e)
11:        f ← projectOntoUnitSphere(f)
12:        t1 ← triangle(a, d, e)
13:        t2 ← triangle(b, d, f)
14:        t3 ← triangle(c, e, f)
15:        t4 ← triangle(d, e, f)
16:        V* ← V* ∪ (d, e, f)
17:        T* ← T* ∪ (t1, t2, t3, t4)
18:    end for
19:    return (V*, T*)
20: end function
21:
22: function GEODESIC(resolution)
23:    (V, T) ← setIcosahedron()
24:    i ← 1
25:    while i ≤ resolution do
26:        i ← i + 1
27:        (V, T) ← SubDivide(V, T)
28:    end while
29:    return (V, T)
30: end function
```

5.2.2 Prismatic Device

Actions are performed on a single point on a kinematic device. However, there is no need to actually fix *parameters* and *variables* of the device to specific points. If, for instance, the prismatic device is at its first limit, the position of said limit can be any part of the actual object, e.g. the lowest or highest part. This is consistent as long as always the same part is chosen in order to identify critical points. However, it is sufficient to identify device properties relative to the point, where the *action* is performed. In case of the prismatic device, this allows us to drop the *parameter limit* altogether when considering discretization.

The remaining parameters are the translation *axis* $(x_{axis}, y_{axis}, z_{axis})$ and the *maximum displacement* d_{max}. The *axis* is dealt with according to Section 5.2.1. d_{max} and the only variable, the *current displacement* d, are scalars. For both, the same limit and number of partitions n is chosen.

Let S_{prism} be the set of discrete states for the prismatic device, then $s_{prism} = (x_{axis}, y_{axis}, z_{axis}, d_{max}, d) \in S_{prism}$ is a corresponding state value. The size of the state space is $n \times n \times an$, where an is the number of vertices from the discretized sphere. States with $d > d_{max}$ can be ignored and their probability is set to zero. Other state probabilities are initialized uniformly.

5.2.3 Static Device

The only *parameter* of the static device is the *position* $(x_{pos}, y_{pos}, z_{pos})$. The position has no influence on the device's behavior and can be dropped. The same argument as for the prismatic device holds true here. This results in a zero-dimensional state space and only the *type* itself is assigned a probability.

5.2.4 Revolute Device

Let us assume that the revolute device is round, disk-like and symmetric in such a way that there is no need to distinguish between different rotation angles α. Since the *radius* r is again only depending on the current point the *action* is applied to, it can be dropped, too. The remaining parameters are the location of the *center* $c = (x_{center}, y_{center}, z_{center})$ and the rotation *axis* $a = (x_{axis}, y_{axis}, z_{axis})$.

For the *axis* see Section 5.2.1. The *center* is a 3D-coordinate. The belief is formed over a box and can be discretized by considering the three dimensions as individual parameters.

For each of them, the limits and number of partition n are equal and together form a grid.

Let $S_{revolute}$ be the set of discrete states for the prismatic device, then $s_{revolute} = (x_{axis}, y_{axis}, z_{axis}, x_{center}, y_{center}, z_{center}) \in S_{revolute}$ is a corresponding state value. The state space is $n \times n \times n \times an$, where an is again the number of vertices from the sphere. State probabilities are initialized uniformly.

5.3 Transition Model

The following sections explain how kinematic devices react to a given action. This section basically describes the *transition* model used in this work. The objective is not to exactly simulate the behavior, but to approximate it efficiently. Each device is modeled by a function that returns a vector, which represents the translation in 3D space and a quaternion to indicate its rotation.

5.3.1 Prismatic Device

Listing 5.1 shows the source code for simulating a prismatic kinematic device. The lock's parameters are given by the vectors *limit1*, *limit2*, and *direction*. They correspond to the parameters mentioned in Chapter 5. The *direction* vector is actually redundant, since it can be calculated using the two limits. The only device variable, the displacement, corresponds to the vector *shift*.

We assume that only the amount of force in lock direction is important. Therefore, the actual point where the *force* vector is applied can be ignored. The force in lock direction is calculated by projecting the *force* vector onto the *direction*. If its magnitude is smaller than the given stiction, no movement occurs. Otherwise, the force directly leads to a translation along the axis, which is restricted by the two limits. If a limit is reached, the movement is reduced accordingly.

Since the prismatic lock only allows for translational movement, the rotation is set to zero.

Listing 5.1 Source code for simulating a prismatic lock

```
std::pair<arr,ors::Quaternion> prismatic_simulate(arr force, arr limit1, arr
    limit2, arr shift, double stiction, arr direction){
  arr translationt(3);
  arr forceInLockDirection(3);
  forceInLockDirection = double(~force * direction) * direction;
  double appliedForce = sqrt(~forceInLockDirection * forceInLockDirection);
  if (appliedForce > stiction){
        translation = 1.0 * forceInLockDirection;
        int i = 0; //dimension index
        for (int j=1;j<3;j++){
              if(direction(j)>direction(i)){
                    i = j;
              }
        }
        if(shift(i) + translation(i) > limit2(i) - limit1(i)){
              //limit2 reached
              translation = limit2 - limit1 - shift;
        } else if((shift(i) + translation(i) < 0 && limit1(i) < limit2(i)) ||
            (shift(i) + translation(i) > 0 && limit1(i) > limit2(i))) {
              //limit1 reached
              translation = -shift;
        }
  } else {
        movement = {.0,.0,.0};
  }
  ors::Quaternion rotation;
  rotation.setZero();
  return std::make_pair(movement,rotation);
}
```

5.3.2 Revolute Device

Listing 5.2 shows the source code for the simulation of a revolute lock. The lock parameters from Chapter 5 are the rotation *axis* and the *center* as vectors. In contrast to the prismatic lock, the actual point, onto where the *force* vector is applied, is necessary.

In order to obtain the angle α on which to rotate along the axis, the system is shifted by the *center* and the *point* is projected onto the plane which can be defined by using the *axis* as normal. The length of the distance from the *center* towards the *point* is the *radius*. The *force* vector is projected onto the same plane and the components orthogonal to the

Listing 5.2 Source code for simulating a revolute lock

```
std::pair<arr,ors::Quaternion> revolute_simulate(const arr& force, const arr&
    point, const arr& axis, const arr& center, double stiction){
  arr translation(3);
  ors::Quaternion rotation;
  arr point_ = point - center; //center point towards origin
  double projection_length = scalarProduct(point_, axis); //calc radius (point
      to center) on projection plane
  arr radius_vec = axis;
  radius_vec *= projection_length;
  radius_vec -= point_;

  double radius = length(radius_vec);
  double sign = -1.0; //rotation angle should be in direction of force,
      independent of (negative) axis direction.
  if(projection_length < 0.0){
        sign = 1.0;
  }

  if(radius < 0.001){
        translation = 0.;
        rotation.setZero();
        return std::make_pair(translation,rotation);
  }

  arr force_direction = crossProduct(axis,radius_vec);
  force_direction /= length(force_direction);
  arr force_applied = scalarProduct(force, force_direction) * force_direction;
  double force_value =length(force_applied);

  if(force_value > stiction){
        double alpha = sign * (MLR_2PI/8) * force_value / radius;
        arr point_rotated(3);
        rotation.setRad(alpha,axis(0),axis(1),axis(2));

        point_vec = rotation * point_vec; //rotate
        point_rotated(0) = point_vec.x;
        point_rotated(1) = point_vec.y;
        point_rotated(2) = point_vec.z;*/

        point_rotated = rotation.getArr() * point_;

        translation = point_rotated - point_;
        return std::make_pair(translation,rotation);
  }

  translation = 0.;
  rotation.setZero();
  return std::make_pair(translation,rotation);
}
```

Algorithmus 5.2 Belief Update Algorithm

```
1:  function BELIEFUPDATE(b,a,o)
2:      b* ← Zeros(b)
3:      for all s ∈ S do
4:          s* ← TransitionModel(s, a)
5:          for all s' ∈ S do
6:              b*[s'] ← b*[s'] + b[s] · N(s* − s', σ_trans)
7:          end for
8:      end for
9:      for all s ∈ S do
10:         o* ← ObservationModel(s, a)
11:         b*[s'] ← b*[s'] · N(o* − o, σ_obs)
12:     end for
13:     return Normalize(b*)
14: end function
```

projected *point* are considered. The length of the *force* vector *force_value* along with the radius determine the angle in force direction:

$$\alpha = \frac{2\pi}{8} \frac{force\_value}{radius} \tag{5.1}$$

After rotating the point by α around the *axis*, the resulting translation vector and the rotation quaternion are calculated. If however, the *force_value* is smaller than the stiction, they are set to zero. The same is true, if the considered point is in direct proximity to the axis.

5.4 Belief Update

The belief update is performed according to Equation (4.8). The old belief b is updated in Algorithm 5.2 using the last action a and observation o [BKL14].

Line 2 initializes the new belief b^* with the same state size as b. In lines 3-4 for each possible state s, the expected next state s^* is calculated by using the *transition model*. Lines 5-6 updates b^*. The transition probability to go from s to s' is proportional to the density of a Gaussian distribution with standard deviation σ_{trans} at the distance $s^* - s'$ between the considered state s' and the expected state s^*. In lines 9-11 o is compared to the expected observation o^* from the *observation model*. This influences b^* using the

density of a Gaussian distribution with standard deviation σ_{obs} at the distance $o^* - o$. Line 13 normalizes the belief so that the summed probability over the state space is one.

We defined an observation o to be a combination of a translation vector t and a quaternion Q for the rotation. Let the following equation be our measure for the observation distance:

$$o^* - o = \omega_r \cdot \|t^* - t\| + (1 - \omega_r) \cdot (1 - |Q^* \cdot Q|), \tag{5.2}$$

where $\omega_r \in [0, 1]$ is the rotation weight, $\| \cdot \|$ denotes the Euclidean norm (or 2-norm). $(1 - |Q^* \cdot Q|) \in [0, 1]$ is a distance measure for quaternion rotation [Huy09]. $Q^* \cdot Q$ is the quaternion inner product. For a rotation of $180°$ the measure returns one, for a rotation of $0°$ zero.

In line 5 the algorithm iterates over every possible state. Since we assume only *variables* x change if actions are applied, it is sufficient to iterate over states that share the *type* m and *parameters* θ. This additionally allows to efficiently calculate the distance $s^* - s'$ as distance between *variables* $x^* - x'$. The only variable comes from the prismatic device and is the scalar *current displacement* d, which can be translated into a corresponding displacement vector by multiplying it with the *rotation axis* a. By combining this with a quaternion with zero-rotation, we can apply the distance measure from Equation (5.2) as for the observations.

5.5 Active Learning

In Section 4.3, we introduced two different action selecting equations, the maximization of expected entropy (4.11) and of the *expected cross entropy (MaxCE)* (4.12). Both strategies require to calculate for every action an expected value over the possible observations, given the current belief. The Monte-Carlo method has been used to solve this problem in Algorithm 5.3[BKL14].

In line 2 we iterate over all possible actions. We defined actions to be continuous over *positions* p and *force vectors* v. The simulation is not combined with any real object with a surface, therefore the *positions* can be an arbitrary element of the Euclidean space $p \in \mathbb{R}^3$. We discretize p the same way as the *center* c in Section 5.3.2. Let us fix the magnitude m of v, such that the space of v becomes a sphere. We can then apply the strategy from Section 5.2.1.

Algorithmus 5.3 Active Learning Algorithm

```
1: function ACTIVELEARNING(b,k)
2:     for all a ∈ A do
3:         Hsum[a] ← 0
4:         for all i ∈ [1, k] do
5:             s ∼ b
6:             s′ ∼ Gaussian(TransitionModel(s, a), σ_trans)
7:             o ∼ Gaussian(ObservationModel(s′, a), σ_obs)
8:             b′ ← BeliefUpdate(b, a, o)
9:             Hsum[a] ← Hsum[a] + H(b, b′)
10:        end for
11:    end for
12:    return argmin_a Hsum
13: end function
```

For each action k samples are drawn from the observation space. Line 5-8 describe how a new belief is sampled, by first sampling a state s from the current belief b. The next state s' is obtained by invoking the transition model and adding Gaussian noise with standard deviation σ_{trans}. The observation model gives the observation o. Again Gaussian noise with σ_{obs} is applied. Given b, o, and a a new belief b' is calculated. From this, we calculate $H(b, b')$, which is the information measure. For the *entropy* case $H(b, b') = H(b')$ holds true, where $H(b')$ is calculated according to Equation (4.10). In the *MaxCE* case $H(b, b') = -H[b, b']$ is calculated according to Equation (4.13). In Line 9 the information measures for each of the k samples are summed. Line 12 returns the most informative action according to the used measure.

5.5.1 Comparison Between Minimizing Entropy and Maximizing Cross Entropy

Minimizing the Entropy only considers the information of the augmented posterior. In case of a wrongly biased belief, this may lead to situations where in terms of entropy, it may be beneficial to actually strengthen this belief instead of discarding it. Maximizing Cross Entropy on the other hand favors changes of the model posterior in any direction, regardless of the entropy [KLT14].

Figure 5.2 shows an initial, posterior belief and the expected posteriors after taking three different actions. The state space consists of five states, where the last four states share a common type, while the first one differs. This setup is a simplification of our actual system. Instead of the three device types, we use two. Type one corresponds to

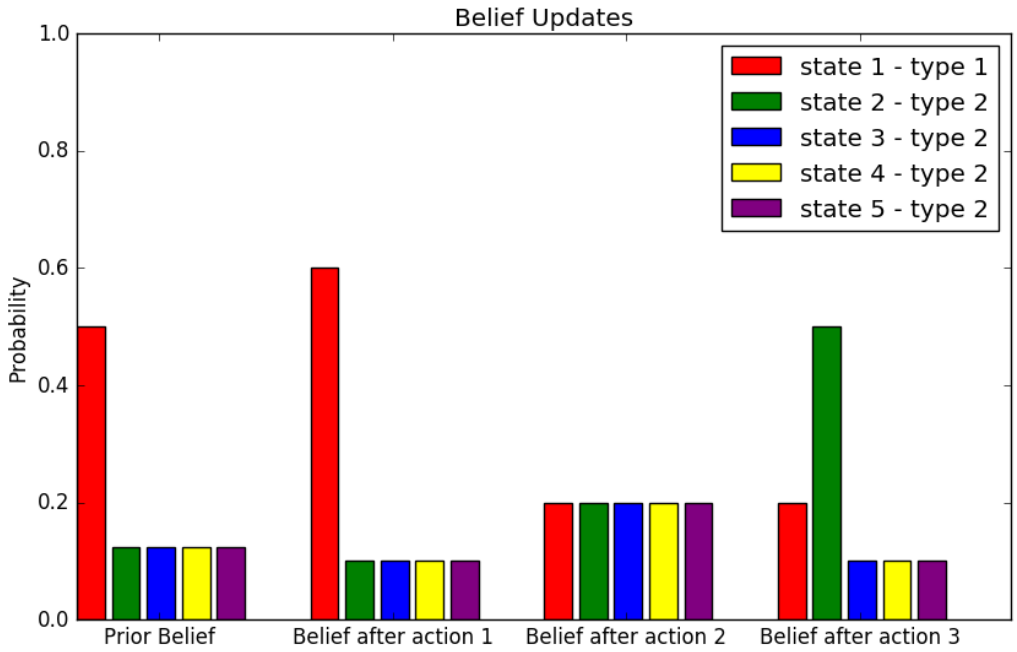

Figure 5.2: Example for possible belief updates.

the *static* device, type two to the state space reduced *prismatic* device. The prior belief is the actual initialization, where each type has the same probability. The three actions are arbitrary, while the augmented posterior beliefs are the important part. For action one, the belief in type one is increased. Action two decreases the belief in type one, while equally strengthening it for every state of type two. Action three decreases the belief in type one and strengthens a specific state of type two.

Table 5.2 is a comparison for the two information measures given the same system. This simple example shows, how drastically the choice of measure can change the selected action. In our actual case, the bias towards the *static* model is even greater, since the state space of both other devices is significantly higher. So the incentive to strengthen the belief in the *static* model should be even higher for the entropy based selection.

Belief	Entropy	Cross-Entropy
Prior	2.0	-
after executing action 1	1.77	2.03
after executing action 2	2.32	2.32
after executing action 3	1.96	2.53

Table 5.2: Comparison of Entropy and Cross Entropy based Active Learning techniques. The action that is chosen according to a given measure is marked red.

6 Evaluation

6.1 Belief Update

This experiment aims to show how the belief update works. Most importantly, it will indicate that even though no movement is detected, the belief is updated correctly and reduces the state space we have to consider.

The parameters for this experiments are the following: The stiction is set to 0.1. We assume σ_{trans} and σ_{obs} to have the same value: $\sigma_{trans} = \sigma_{obs} = 0.01$. The rotation weight for the distance measure is defined as $\omega_r = 0.5$.

Two dimensions of the discretized *prismatic* state space, d and d_{max}, have 15 bins each and a range form zero to one. For the *axis*, the resolution after two subdivisions is used, which equals the geodesic sphere with 312 vertices. For the *revolute* state space, the axis is the same. The *center* parameter is divided into 15 bins for each dimension. Limits are from -1 one to 1.

The five executed actions are shown in Table 6.1 and the force direction in Figure 6.2. The resulting Belief State is demonstrated by Figure 6.1. Its axis are the same as in Figure 6.2.

The forces for the first and the second action are directed in opposite directions. Since both test the same axis, the difference between the beliefs is only marginal. The third force applied tests another direction. Since for none of them movement is detected,

action number	force vector	force amount
1.	$(-0.036, -0.111, -0.058)$	0.13
2.	$(0.036, 0.111, 0.0581)$	0.13
3.	$(-0.094, 0.068, 0.0581)$	0.13
4.	$(0.0, 0.5, 0.0)$	0.5
5.	$(-0.116, 0.0, -0.0581)$	0.13

Table 6.1: Actions used to demonstrate the Belief Update.

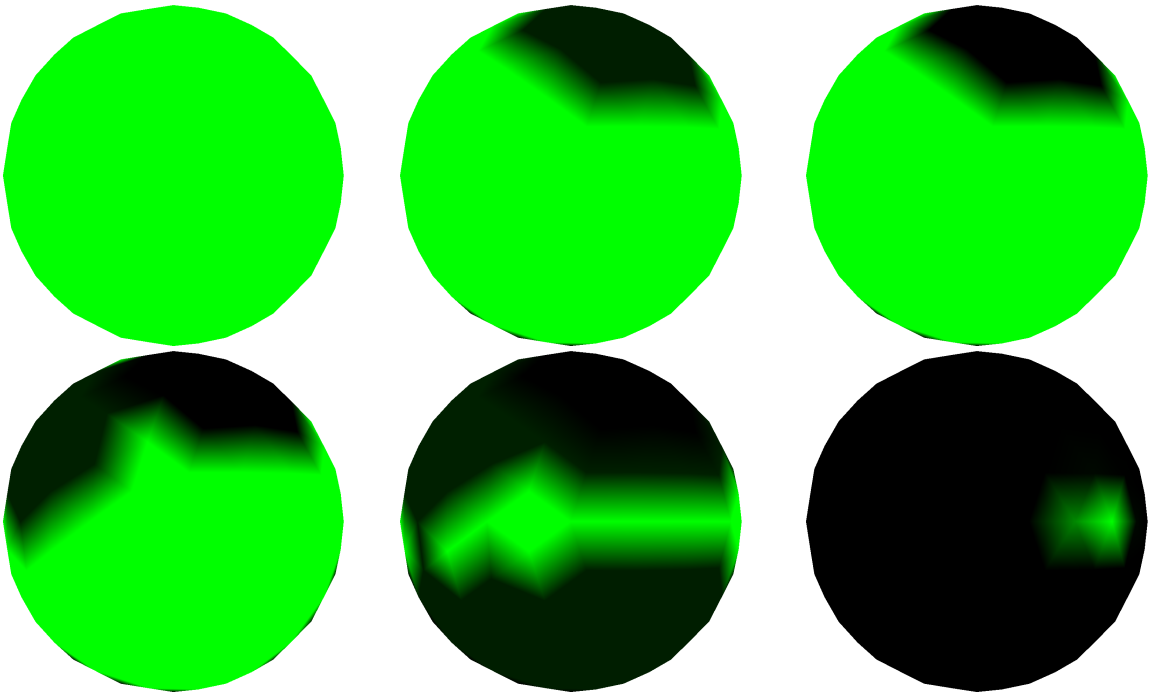

Figure 6.1: The spheres show the Belief in a given *axis* by integrating over all other *parameters* and *variables*. The colors are scaled in such a way, that the highest value is assigned a constant green, regardless of its actual value. Since the spheres are nearly symmetric, it is sufficient to show one side only. The beliefs show the updates from left to right and from top to bottom.

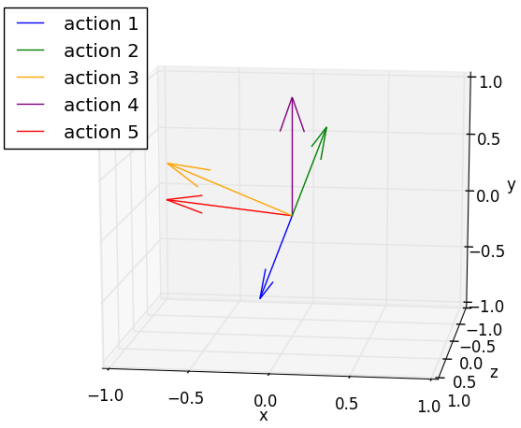

Figure 6.2: Direction of the force applied as actions for the belief update. In order to give a better impression on the actual direction in 3D space, the length of the force vector is normalized to one. The axis are located as in Figure 6.1

model	parameter θ	variables x
prismatic	$(x_{axis}, y_{axis}, z_{axis}) = (0.8944, 0.0, 0.4472)$ $d_{max} = 0.3162$	$d = 0.0316$
static	-	-
revolute	$(x_{center}, y_{center}, z_{center}) = (0.2, -0.1, 0.6)$ $(x_{axis}, y_{axis}, z_{axis}) = (0.0, 0.8944, 0.4472)$	-

Table 6.2: Experimental parameters and variable initialization for the three device types.

only the belief in direct neighborhood is reduced. The fourth action uses a higher force amount, therefore, even though again no movement is detected, the effected neighborhood is increased. For the fifth action, movement is detected. This leads to a very specific belief in the axis, which corresponds to the locks ground truth.

6.2 Active Learning strategies

We evaluated the different action selecting strategies by conducting multiple experiments. For each type, the *prismatic*, *static* and *revolute* device, we tested the three strategies, *random*, *entropy*-based, and *MaxCE*. For each combination of type and strategy, we conducted 50 trials with five action selections each and measured the probability of each type by summing over its *variables* and *parameters*.

The devices' parameter and variable initializations are listed in Table 6.2. The stiction is set to 0.1. The magnitude of the force m is set to $m = 0.15$. We assume σ_{trans} and σ_{obs} to have the same value: $\sigma_{trans} = \sigma_{obs} = 0.01$. The rotation weight for the distance measure is defined as $\omega_r = 0.5$.

d and d_{max} have five bins each and a range form 0 to 1. For the *axis*, the lowest resolution which equals the Icosahedron with 12 vertices, is chosen for the *prisamtic* and *revolute* model. The *center* parameter is divided into five bins for each dimension. Limits are from -1 to 1.

The random strategy chooses an action by sampling from a uniform probability distribution over the discrete set of actions.

Figure 6.3 shows the results for the *prismatic* device, Figure 6.4 includes results for the *static* device, and Figure 6.5 for the *revolute* device.

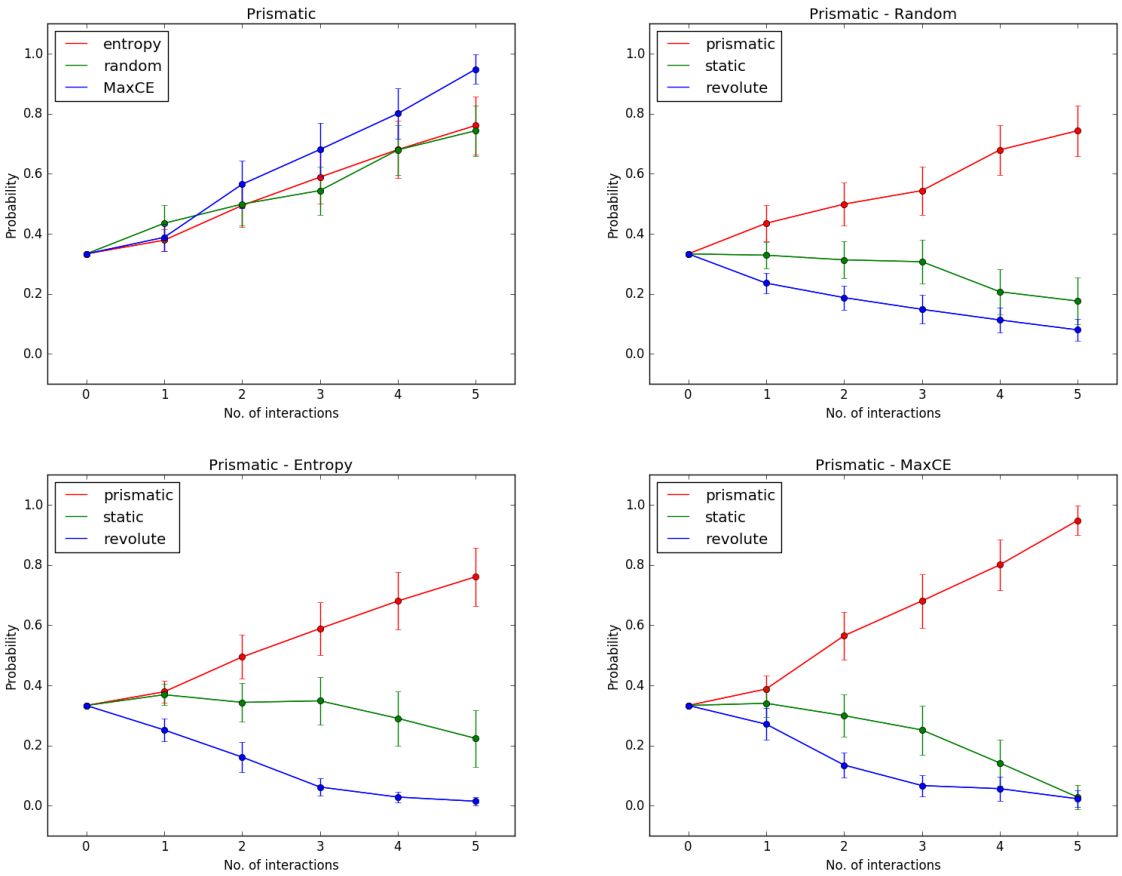

Figure 6.3: Results of the simulation experiments on the *prismatic* device. The graphs show the mean probability for a certain device type. Error bars reflect a 95% confidence interval of the mean estimator.

The top left graph in Figure 6.3 shows the probability over the number of actions for the *prismatic* device for the three different action selecting strategies. While the *entropy*-based strategy barley outperforms the random strategy, *MaxCE* is clearly superior to both of them. Comparing the belief in the *static* model, we notice that the *entropy*-based approach is preserving it the most. This is likely due to the problem mentioned in Section 5.5.1.

Figure 6.4 shows that for the *static* device, the *entropy*-based and the *MaxCE* approach slightly outperform the random strategy. The latter strategy fails to efficiently reduce the belief in the *prismatic* device. This is even more important, since the belief in the *revolute* model is lowered quite fast, regardless of the exploration strategy. The reason is, that in order to disprove the belief in the *prismatic* lock, the force has to be applied in a very specific direction. Both planned strategies are able to do that. Since the initialization

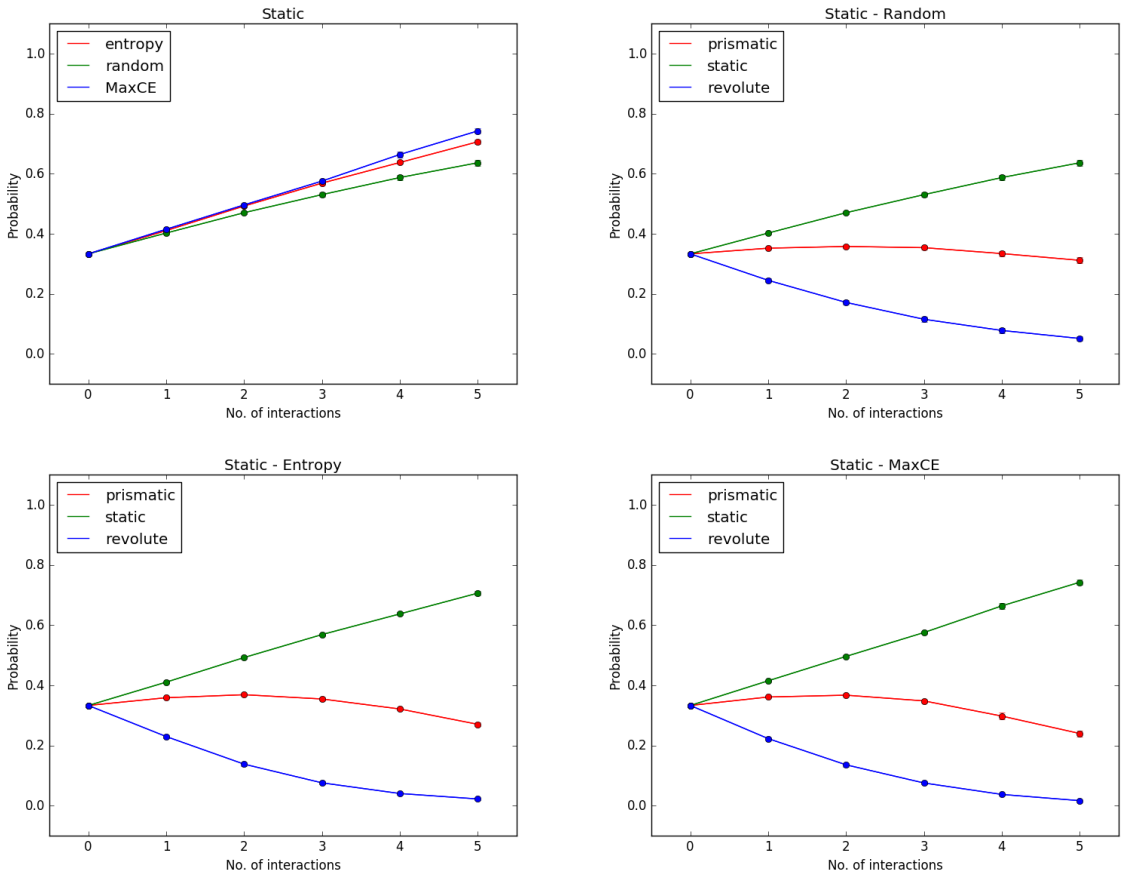

Figure 6.4: Results of the simulation experiments on the *static* device. The graphs show the mean probability for a certain device type. Error bars reflect a 95% confidence interval of the mean estimator.

does not wrongly bias the belief, but actually does so correctly, the difference between those strategies is only marginal.

Figure 6.5 shows the results for the *revolute* device. Again, there is no significant difference between the *MaxCE* and *Entropy*-based approach. Both outperform the random strategy in terms of increasing the correct belief faster. However, after executing the fifth action, the belief is almost equal for all three approaches, since the planned strategies stagnate in increasing the correct belief. The reason for this is simple: The ground truth of the simulated device is not part of the discrete state space of the *revolute* device. For the same reason, after the fourth action, the correct belief at the *entropy*-based strategy, is actually decreased. For a few trials, a behavior is observed that actually can be better explained by the *prismatic* model. Therefore, this problem is due to the low resolution of the discrete state space.

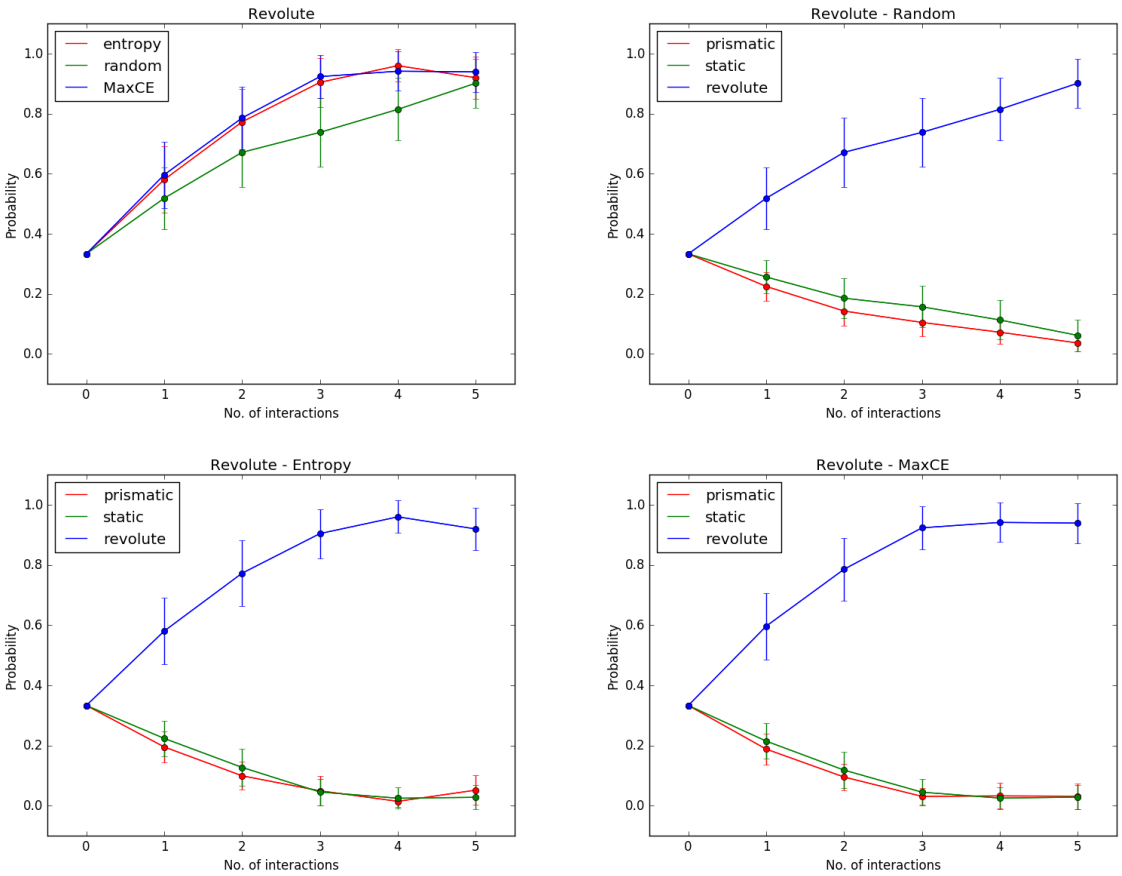

Figure 6.5: Results of the simulation experiments on the *revolute* device. The graphs show the mean probability for a certain device type. Error bars reflect a 95% confidence interval of the mean estimator.

The experiments clearly state, that the *MaxCE* approach is best suited for this scenario. For two locks it is equally good as the original *entropy* based strategy, but clearly outperforms it in case of the *prismatic* device. Both planned strategies are superior to the random strategy.

Calculating the best action takes about 34 seconds. The experiments were conducted on a machine with an Intel(R) Core(TM) i7-3770 CPU @ 3.40G runnning Ubuntu.

7 Discussion

7.1 Results

In the evaluation, we have shown that our approach is capable of identifying the correct device *type* and *parameters* located arbitrarily in the 3D space. Independent of the sequence of actions, the belief update was accurate. Even in cases where no movement was observed, we were able to limit the belief to specific states, where the used actions would have lead to the observed effect. The belief update itself is performed quite fast and even for very high state spaces with a high resolution in terms of discretization the update takes merely seconds. We were able to successfully take the general approach from Barragán, Kaelbling, and Lozano-Pérez [BKL14] from the 2D to the 3D space by introducing novel discretization strategies.

Additionally, we implemented two different Active Learning strategies. We showed that the *entropy*-based approach from Barragán, Kaelbling, and Lozano-Pérez [BKL14] is outperformed by the better strategy introduced by Kulick, Lieck, and Toussaint [KLT14]. This is especially true for scenarios where different device types have large differences regarding the state-space size. Both strategies however share a common problem. The computation time drastically increases with the state-space. In fact, choosing a single action given a current belief on the resolution level of Section 6.2 takes about 34 seconds independent of the chosen strategy, which is not real-time. One solution could be to run the problem on a more powerful platform. The problem that an increased state-space size leads to much higher computation times will persist however. This is especially significant in the 3D space, where the number of parameters is naturally higher, compared to the more simple alternative in the 2D space. In Section 7.2 we discuss some ideas that may solve this problem.

7.2 Alternatives

There are many things which could have been done differently in this work, from the choice of parameters to considering a whole new strategy. In this section, a few of the choices are questioned and alternatives are presented.

7.2.1 Optimizing the current Approach

We assume our strategy is in general a good choice. Since it achieves most of the objectives, it very well might be. One thing to notice is the fact that for the *revolute* device, the parameter state space is actually too large. In fact, several states are essentially the same and are therefore redundant. The two parameters are the *center* and the *axis*. Consider two different states with the same *axis* and *centers* lying on the same line parameterized by the *axis*. Their behavior is identical, therefore they can be considered equal. The same is true for two different states with the same *center*, but opposite *axis* direction. For the *revolute* device, it is actually sufficient to consider only a half sphere.

The first insight however is harder to exploit. In our current model, we discretized the state space. There is no way to simply cut out a whole dimension and calculating for every combination of *center* and *axis*, whether they are identical is cumbersome and the improvement probably insignificant. An easy way to make its simulation more efficient is to discard the limits. The state space would be significantly reduced by two dimensions: The parameter for the *maximum displacement* and the variable for the *current shift*. This would leave us with the *axis* as the only parameter and additionally allows us to consider only half of the sphere. Since the sphere is already discretized in a very efficient way, this would significantly improve the overall performance. However, the device would lose most of its characteristics. The simplification of the problem makes the simulation more unrealistic and not consistent with the real world. Instead of modeling limits, we would have to consider behaviors like a bar falling out of its mechanics in order to approach the real problem again.

Theoretically, even the complex model of the *prismatic* device might have states that actually model the same thing. This would be the case, if the *limit*, the *current displacement*, the *maximum displacement*, and the *axis* are aligned for two states in such a way that the axis are opposed and the two actual limits of each device interchange, while the *current displacements* lead to the same position of the device. Since one would probably need to construct a specific discretization in order to enable this scenario, there is no need to further investigate this opportunity.

7.2.2 Altering the current Approach

An alternative to the discretization is to use continuous probability functions for the belief. It would be desirable to have such a function for each device *type*. This could enable an analytical solution for the belief update problem. However, it requires for each *type* to find probability distributions for the prior belief that are conjugate prior

for the likelihood function. Since *type* beliefs are significantly different, it is a difficult problem. One solution would be to use a multidimensional Gaussian distribution as prior. Instead of a specialized belief over a sphere for the direction, this approach would use the whole euclidean space. The Von Mises-Fisher distribution even allows to model a probability function directly on a sphere [Fis53]. Since both probability distributions allow only one maximum, they will not be sufficient to model the belief. Another idea is to combine the advantages of a continuous distribution and the discrete alternatives. Some parts of the belief can be modeled continuously, if a sufficient function is found.

The current transition model is based on a simulation which is only approximating the real world. These simulations may therefore be inaccurate and if our approach is tested on the PR2, the belief update and the Active Learning strategies may fail for this reason. Barragán, Kaelbling, and Lozano-Pérez [BKL14] used a physic simulation library (Bullet Physics Library). Appropriate models have to be constructed in order to use it. The advantage is that the resulting behavior is more accurate. Another approach is to parametrize the simulation functions. These parameters can then again be learned from data, resulting in a very accurate prediction. For this, a new optimization problem needs to be solved.

One of said parameters would be the device's stiction. Currently we assume that its value is known. By testing we can actually measure it. Alternatively, we can add the stiction as a parameter to the belief. This however increases the state space further. Given the computational problems already present, this seems like a bad idea.

Simulating the device using a proper physic simulator has an additional effect of allowing to easily define new types of actions. Currently, pushing is the only way of manipulation. Given a proper model of the device, it is possible to add more sophisticated actions like grasping and pulling, without having to deal with the problem of writing a specific simulator by hand.

One of the significant differences between this work and the work of Barragán, Kaelbling, and Lozano-Pérez [BKL14] is the considered space. While the latter considers the problem in the 2D space, we work in the 3D space. The disadvantage of this approach is higher number of parameters and variables, leading to a more complex computation and longer calculation times. Given a good model of the considered device, it is possible to reduce the size of the state space. Certain states can be marked as incorrect due to the knowledge of the device. It is for example unreasonable to assume the *center* of a *revolute* device to be in the air. As long as the device is not extraordinary, it is save to assume that the *center* is in fact located within the body. An especially useful opportunity the model would grant is the possibility to assume that a device is operated on a surface. If one could find a corresponding 2D plane, the assumptions from Barragán, Kaelbling, and Lozano-Pérez [BKL14] hold true. We could discard the 3D strategy and reduce the state space significantly.

Another benefit of the model is related to the Active Learning strategies. It is possible to limit the number of considered actions. Since performing actions like pushing is only feasible on the surface of an actual object, actions which do not meet this obvious requirement can be ignored. Additionally, using tools like KOMO [Tou14] that allow to plan and calculate movements, we can filter out unreachable or unfeasible actions as well. For each action, we can plan and check whether the PR2 robot will be able to reach the desired position. If, for instance, the path to a position is blocked or the desired force vector can not be applied due to motion restrictions, there is no need to consider the action. We can discard it before calculating the action's outcomes. Another way to reduce the number of actions is to simply sample them in such a way that their direction and expected results differ. In doing so, the number of considered actions could be reduced to such a small amount that an analytical solution, instead of a Monte-Carlo simulation, becomes feasible. Instead of sampling locks from the current belief, every discrete state can be considered. The actual expected entropy or cross-entropy can be calculated.

Having a functional model of the lockbox and its devices, which preferably allows to simulate interactions with the PR2, would greatly benefit this work. It would allow us to limit the number of feasible actions, while increasing their variety. Potentially this would allow us to reduce the state space for certain beliefs. Additionally, the transition model would be more accurate and easily applicable to the real world.

7.2.3 Alternative approaches

Sturm, Stachniss, and Burgard [SSB11] propose a strategy that is fundamentally different compared to our work. Instead of considering a set of given devices and learning how to manipulate them, the work interprets the world as a kinematic graph. Object correspond to vertices, edges model their kinematic relationship. This allows to describe more complex relationships. Objects are not limited devices but can stand in arbitrary relationship to one another. The graph is learned using a probabilistic framework. It relies on observations to learn how objects restrict each other and through which joints they are related.

Additionally, the learned models can be generalized and transfered to other, similar problems. The advantage of this work is the flexibility it has in describing different possible scenarios. It can not explain, how to actually generate observations that leads to information gain. Combining this with an Active Learning strategy would make this approach comparable. This work could be combined with ours in a meaningful way. Using the model and Active Learning strategies presented, we could generate the observations and data which are necessary for building the kinematic graph. We can than use the graph as a starting point for more complex tasks.

Similar to [SSB11], the work of Martín and Brock [MB14] or Katz and Brock [KB08] could be used to learn joints. Most of the work presented in Chapter 2 identifies kinematic devices only if motion is preexistent. One significant problem is to actually generate this movement. The solution from Barragán, Kaelbling, and Lozano-Pérez [BKL14] allows to gather information regardless. The absence of movement can be used to update the belief.

Narayanan and Likhachev [NL15] introduce an action selection strategy that incorporates a goal state. This approach seems useful if we want to learn device parameters while simultaneously learning joint dependencies. In Section 1.2, we describe how our work serves as a basis for joint dependency learning. The goal state for the joint identification part could be the desired action from the joint dependency discovery with the objective to lock or unlock the joint. For this however, the higher level would need to cope with uncertainty over joint parameters, which is not the case at the moment. Additional problems are that Narayanan and Likhachev [NL15] assume every part of the mechanism to be observable, a deterministic transition model for a given device, and initial movement to reduce the number of candidates for the belief planner.

Katz, Pyuro, and Brock [KPB09] use Reinforcement Learning to learn actions for a given relational state representation. The learned knowledge can be transfered to previously unseen objects. If we would consider mechanisms with multiple joints, this work could help to reduce the number of necessary actions. Currently, we consider kinematic devices that only have one joint. Therefore, learning actions based on a relational state representation is unnecessary.

Hausman et al. [HNOS15] use a different model for the belief. Instead of a discrete state space, they use a particle filter. This approach is better suited for the larger state space of kinematic devices in 3D space. In retrospective, it could have been more effective to use this approach. They were able to successfully conduct experiments and identify different joint types using the PR2.

7.3 Insights

Originally, the idea of this work was to use KOMO [Tou14], a constraint optimization framework, to calculate feasibility and costs for certain actions on the lock box. We would use this information to create an additional reward function for the approach presented in [KOT15b] to solve the lockbox problem while taking into account cost for motion. During the course of this work, it became clear that in order to achieve this, we first need to find a way to reliably identify joints and learn their parameters,

before we could apply the strategy. As a result, the focus of the work shifted to *kinematic identification*.

At first, ideas involved learning this information from trajectories, similar to the work of Sturm, Stachniss, and Burgard [SSB11]. The main problem is, that actual motion needs to be generated first. Instead of identifying joints based on motion, the objective became to execute actions on mechanisms and update a belief over its types and parameters, even if no motion could be detected. One of the parameters we wanted to find was the direction of the prismatic joint or the rotation axis for the revolute joint. A belief over an axis or direction is basically a belief over a sphere in 3D space. One way to model this belief is to use a multidimensional Gaussian distribution over the euclidean space with its mean fixed at the origin and only consider the cut with the unit sphere. Due to symmetry, we would then be able to execute a push action on the mechanism. If no movement was detected, we would then tilt the Gaussian away from the force direction. The way we have to manipulate the covariance matrix in order to correctly update the belief, which is the Gaussians cut with the sphere, is however unclear. The cut with the unit sphere would allow to model a uniform distribution or two symmetrical maxima. With a single Gaussian distribution, it is not possible to model more maxima, which is insufficient for our belief. The same is true for the Von Mises-Fisher distribution [Fis53]. Instead, we would have to use multiple Gaussians in order to get the desired flexibility, making the update process even more complicated. The approach presented in this thesis simply discretized the belief. This allows additionally to combine the belief over the sphere with other parameters like the location. Finding a distribution over a sphere that allows for the same flexibility as the discrete version would potentially allow a continuous solution of the problem. As mentioned in Section 7.2.2, finding a conjugate prior to this function would even allow for an analytical update and would simplify the whole problem further.

7.4 Conclusion

We implemented a method that is able to correctly identify the *types* and *parameters* of kinematic devices in 3D space. It involves an efficient way to discretize the belief space, which is updated using a Bayes Filter, that allows to extract useful information, even if actions yield no movement. An Active Learning action selection is used to reduce the number of actions necessary for the identification. We showed that using plain entropy as information measures is outperformed by the *MaxCE* strategy.

What we weren't able to achieve is a real-time calculation for the action selection. We want to enable the PR2 to autonomously work on the lockbox. A calculation that takes more than we few seconds is unsatisfactory. If we can solve this problem, we can test

our method outside of simulations. This will tell us, how sophisticated our transition model actually is. Finding the correct parameters for this model is another challenge we have to tackle.

A Zusammenfassung

Diese Arbeit handelt von der Identifizierung kinematischer Objekte als Teil des Lockbox Problems. Dabei handelt es sich um die Herausforderung, einen Roboter in die Lage zu versetzen, eine Box zu öffnen, die durch mehrere in Reihe geschaltete Schlösser verriegelt ist. Um mit diesen Schlössern umzugehen, muss der Roboter die zugrundeliegenden Typen und Parameter ermitteln. Wir lösen dieses Problem, indem wir unseren Glauben (Belief) an diese als diskrete Wahrscheinlichkeitsverteilung modellieren. Dabei wird die Richtungsinformation als Geodätische Spähre diskretisiert, was den Vorteil hat Richtungen isotrop zu verteilen und diese somit nicht zu verzerren. Die Verbesserung des Beliefs wird aufgrund von Erfahrungen, mittels eines Bayes Filters durchgeführt. Dieser erlaubt es, die korrekten Parameter einzugrenzen, auch wenn eine Aktion des Roboters zu keiner sichtbaren Bewegung führt. Die Auswahl dieser Aktionen erfolgt unter Betrachtung des erwarteten, zukünftigen Beliefs und zielt darauf ab, die Anzahl Aktionen die benötigt werden um den Mechanismus erfolgreich zu identifizieren, zu minimieren. Dafür testen wir verschiedene Maße für die Informationen, indem wir sie mit einer zufälligen Aktionsauswahl in einer Simulation vergleichen. Dabei stellt sich heraus, dass das auf der Kreuzentropie basierende Maß, das effektivste ist. Insgesamt waren wir in der Lage, korrekt zwischen *prismatischen*, *statischen* und Geräten mit *Drehgelenk* im Dreidimensionalem Raum zu unterscheiden.

Bibliography

[AKB13] A. M. Auersperg, A. Kacelnik, A. M. von Bayern. "Explorative learning and functional inferences on a five-step means-means-end problem in Goffin's cockatoos (Cacatua goffini)." In: *PloS one* 8.7 (2013), e68979 (cit. on p. 15).

[BKL14] P. R. Barragán, L. P. Kaelbling, T. Lozano-Pérez. "Interactive Bayesian Identification of Kinematic Mechanisms." In: *IEEE Conference on Robotics and Automation (ICRA)*. 2014. URL: http://lis.csail.mit.edu/pubs/barragan-icra14.pdf (cit. on pp. 19, 22, 23, 31, 34, 45, 46, 57, 59, 61).

[CB03] S. Chiappa, S. Bengio. *HMM and IOHMM modeling of EEG rhythms for asynchronous BCI systems*. Tech. rep. IDIAP, 2003 (cit. on p. 25).

[Fis53] R. Fisher. "Dispersion on a Sphere." In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 217.1130 (1953), pp. 295–305. eprint: http://rspa.royalsocietypublishing.org/content/217/1130/295.full.pdf. URL: http://rspa.royalsocietypublishing.org/content/217/1130/295 (cit. on pp. 59, 62).

[HNOS15] K. Hausman, S. Niekum, S. Osentoski, G. S. Sukhatme. "Active articulation model estimation through interactive perception." In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 3305–3312 (cit. on pp. 23, 61).

[Huy09] D. Q. Huynh. "Metrics for 3D Rotations: Comparison and Analysis." In: *J. Math. Imaging Vis.* 35.2 (Oct. 2009), pp. 155–164. URL: http://dx.doi.org/10.1007/s10851-009-0161-2 (cit. on p. 46).

[KB08] D. Katz, O. Brock. "Manipulating articulated objects with interactive perception." In: *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE. 2008, pp. 272–277 (cit. on pp. 21, 22, 61).

[KLT14] J. Kulick, R. Lieck, M. Toussaint. *Active Learning of Hyperparameters: An Expected Cross Entropy Criterion for Active Model Selection*. e-Print arXiv:1409.7552. 2014 (cit. on pp. 22, 35, 47, 57).

[KOT15a] J. Kulick, S. Otte, M. Toussaint. "Active Exploration of Joint Dependency Structures." In: *(ICRA 2015)*. 2015 (cit. on pp. 18, 19).

[KOT15b] J. Kulick, S. Otte, M. Toussaint. *Robots Solving Serial Means-Means-End Problems*. RSS Workshop on Combining AI Reasoning and Cognitive Science. 2015 (cit. on pp. 18, 21, 61).

[KPB09] D. Katz, Y. Pyuro, O. Brock. "Learning to manipulate articulated objects in unstructured environments using a grounded relational representation." In: *Robotics: Science and Systems IV* (2009), p. 254 (cit. on pp. 22, 61).

[MB14] R. M. Martín, O. Brock. "Online interactive perception of articulated objects with multi-level recursive estimation based on task-specific priors." In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 2494–2501 (cit. on pp. 21, 61).

[NL15] V. Narayanan, M. Likhachev. "Task-oriented planning for manipulating articulated mechanisms under model uncertainty." In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 3095–3101 (cit. on pp. 22, 61).

[Oxf16] Oxford University Press. *problem-solving. Oxford Dictionaries.* online. Mar. 2016. URL: http://www.oxforddictionaries.com/de/definition/englisch/problem-solving (cit. on p. 15).

[PWT15] S. Pillai, M. R. Walter, S. Teller. "Learning articulated motions from visual demonstration." In: *arXiv preprint arXiv:1502.01659* (2015) (cit. on p. 21).

[Sha01] C. E. Shannon. "A Mathematical Theory of Communication." In: *SIGMOBILE Mob. Comput. Commun. Rev.* 5.1 (Jan. 2001), pp. 3–55. URL: http://doi.acm.org/10.1145/584091.584093 (cit. on p. 34).

[SJS+10] J. Sturm, A. Jain, C. Stachniss, C. C. Kemp, W. Burgard. "Operating articulated objects based on experience." In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. Oct. 2010, pp. 2739–2744 (cit. on p. 21).

[SRS+05] L. R. Santos, A. Rosati, C. Sproul, B. Spaulding, M. D. Hauser. "Means-means-end tool choice in cotton-top tamarins (Saguinus oedipus): finding the limits on primates' knowledge of tools." In: *Animal cognition* 8.4 (2005), pp. 236–246 (cit. on p. 15).

[SSB11] J. Sturm, C. Stachniss, W. Burgard. "A Probabilistic Framework for Learning Kinematic Models of Articulated Objects." In: *J. Artif. Int. Res.* 41.2 (May 2011), pp. 477–526. URL: http://dl.acm.org/citation.cfm?id=2051237.2051252 (cit. on pp. 21–23, 60–62).

[Tou14] M. Toussaint. *KOMO: Newton methods for k-order Markov Constrained Motion Problems*. e-Print arXiv:1407.0414. 2014 (cit. on pp. 60, 61).

[Vin11] J. Vince. *Quaternions for computer graphics*. Springer Science & Business Media, 2011 (cit. on p. 27).

[WKO92] D. White, J. A. Kimerling, S. W. Overton. “Cartographic and Geometric Components of a Global Sampling Design for Environmental Monitoring.” In: *Cartography and Geographic Information Systems* 19.1 (1992), pp. 5–22. eprint: http://www.tandfonline.com/doi/pdf/10.1559/152304092783786636. URL: http://www.tandfonline.com/doi/abs/10.1559/152304092783786636 (cit. on p. 39).

All links were last followed on May 09, 2016.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Unterschrift:

Stuttgart, 10.05.2016

Declaration

I hereby declare that the work presented in this thesis is entirely my own.
I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations.
Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.
The electronic copy is consistent with all submitted copies.

Signature:

Stuttgart, May 10, 2016