UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de Formació Interdisciplinària Superior

The University of Texas at Austin
Cockrell School of Engineering

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Bachelor Thesis

Robust Door Operation with the Toyota Human Support Robot

Robotic Perception, Manipulation and Learning

Miguel Arduengo García

February 2019

Degrees:
Grau en Enginyeria en Tecnologies Industrials
Grau en Enginyeria Física

Advisors:
Dr. Luis Sentis Álvarez
Dra. Carme Torras Genís

Robots are progressively spreading to urban, social and assistive domains. Service robots operating in domestic environments typically face a variety of objects they have to deal with to fulfill their tasks. Some of these objects are articulated such as cabinet doors and drawers. The ability to deal with such objects is relevant, as for example navigate between rooms or assist humans in their mobility. The exploration of this task rises interesting questions in some of the main robotic threads such as perception, manipulation and learning. In this work a general framework to robustly operate different types of doors with a mobile manipulator robot is proposed. To push the state-of-the-art, a novel algorithm, that fuses a Convolutional Neural Network with point cloud processing for estimating the end-effector grasping pose in real-time for multiple handles simultaneously from single RGB-D images, is proposed. Also, a Bayesian framework that embodies the robot with the ability to learn the kinematic model of the door from observations of its motion, as well as from previous experiences or human demonstrations. Combining this probabilistic approach with state-of-the-art motion planning frameworks, such as the Task Space Region, a reliable door and handle detection and subsequent door operation independently of its kinematic model is achieved with the Toyota Human Support Robot (HSR).

Keywords: *door opening, grasping, handle detection, HSR, learning, perception, robot*

Contents

1 Introduction

Robots are no longer confined to factories and they are progressively spreading to urban, social and assistive domains. According to the International Federation of Robotics, a service robot is a robot which operates semi or fully autonomously to perform services useful to the well being of humans. This development is especially interesting against the background of the increasing importance of the service sector. As our economy relies more than ever on the value created by services, they are the key to future competitive advantage. Analogously to the industrial context, automation through the employment of service robots becomes a means to increase the competitiveness of a service. However, in order to become handy co-workers and helpful assistants, they must be endowed with quite different abilities than their industrial ancestors [73, 79].

Human environments have a number of challenging characteristics that will usually be beyond the control of the robot creator. Among these challenges, dynamic enviroments, real-time constraints, and variations in the size and appearance of the manipulated objects can be pointed out. Everyday tasks people take for granted would stump the best robot bodies and brains in existence today. The research efforts to overcome the current limitations could be divided into three main threads: perception, manipulation and learning.

Robot simulation in controlled enviroments indicates that robots can perform well if they know the state of the world with certainty. Although robots in human environments will almost always be working with uncertainty due to their limited view of a changing world, perceptual systems have the potential to reduce its uncertainty and enable robust autonomous operation. As such, perception is one of this most important challenges that the robotics field faces [33].

When it comes to the robot interacting with the enviroment a lesson is that manipulation is hard. Robots crash against Moravec's paradox, machines can be taught to solve difficult problems but it is really hard to make them do the most simple tasks. Even the most sophisticated robot could be unable to fetch and carry a glass of water. Within perfectly modeled worlds, motion planning systems perform extremely well. Once the uncertainties of dynamic human environments are included, alternative methods for control become important. Control schemes must have real-time capabilities in order to adapt to changes in the enviroment.

Finally, by learning from the natural statistics of human environments, robots may be able to infer unobservable properties of the world or select appropriate actions that implicitly rely on these unobserved properties. Learning can also help address implementation problems. Directly programming robots by writing code can be tedious, error prone, and inaccessible to non-experts. Through learning, robots may be able to reduce this burden and continue to adapt once they have left the factory [33].

In order to accelerate the service robotics research, Toyota Partner Robot has developed a compact and safe research platform, Human Support Robot (HSR) which can be operated in an actual home environment (Figure 1.1).

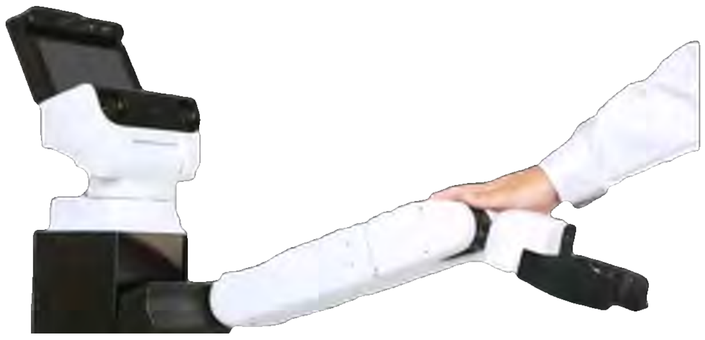


Figure 1.1: Toyota Human Support Robot (HSR) provides life support and assistance

HSR was adopted as the Standard Platform for RoboCup@Home. It was used at the Domestic Standard Platform League (DSPL) for home service robots since RoboCup 2017 Nagoya. Moreover, it has been adopted as a standard platform for the service robot competition of the World Robot Summit (WRS) which is scheduled to be held in 2020 in Japan after the Tokyo Olympic Games [84].

This project aims to contribute to the development of robots that can assist people at home. Service robots operating in domestic environments typically face a variety of objects they have to deal with to fulfill their tasks. Some of these objects are articulated such as cabinet doors and drawers. The ability to deal with such objects is key, for example, to open doors when navigating rooms and to open cabinets to pick up objects in fetch-and-carry applications. The exploration of this task rises interesting questions in robotics fields such as visual perception, object recognition, control algorithms, manipulation under uncertainty, and learning from experience or from humans demonstrations.

In order to correctly operate a door the robot must:

1. Be able to identify doors and their corresponding handle.
2. Grasp the handle.
3. Unlatch the handle.
4. Open the door according to its kinematic model.

In this work, using the Toyota HSR platform, the problem of operating multiple types of doors with a unified framework is adressed. It will be described how progressively, overcoming the challenges that arise in the main threads of service robotics research, the aim of this project is achieved.

In chapter II, the literature regarding the door opening task from different perspectives will be reviewed. In chapter III, it will be presented an approach to identify multiple classes of doors and their corresponding handle as well as their relevant 3D geometric features in real-time. In chapter IV, the main manipulation algorithms used to operate the door taking advantage of the robot sensing capabilities will be studied. In chapter V, a probabilistic framework to learn the kinematic model of doors, that enables the robot to operate unkown doors, learn from experience and human demonstrations. Finally, in chapter VI, the main conclusions of this work will be outlined.

2 Previous Works

The problem of opening doors and drawers with robots has been treated extensively in the specialized literature, in aspects of the task such as the detection of doors and handles [9,37,44,60,68,71,85] and the planning and control of movements for the opening motion [29,34,53,56,59,80]. However, these approaches usually focus either on a particular type of door or in a certain aspect of the whole task and a unified framework for operating different doors completely autonomously has not been proposed.

Regarding the detection of doors and handles, several approaches have been proposed. These techniques incorporate the usage of either images [12,71], or depth data [9,68,85], or both [3,30,37,44,60]. The state-of-the-art in this problem could be discussed through three works: Chen et al. [12] propose a door detection algorithm via convolutional neural network (CNN) for robot autonomous navigation. The proposed algorithm can predict the door poses from a 2D color image of the scene, but does not have any capacity for the detection and segmentation of the handles. Banerjee et al. [6] develop an approacht that enables an ATLAS robot, in a semi-structured environment, as specified by the requirements of the DARPA Robotics Challenge, to detect closed doors. Doors are detected by finding consecutive pairs of vertical lines at a specific distance from one another in a 2D image of the scene. The lines are then recalculated in a 3D space with RANSAC. If there is a flat surface between each pair of lines, it is recognized as a closed door. Handle detection is subsequently carried out by means of color segmentation. This approach makes several assumptions that limit its applicability, since the autors define a specific size, and demand different colors on the door and handle (Fig. 2.1). Llopart el al. [44] describe the implementation of a CNN to extract a Region of Interest (ROI) from an RGB-D image corresponding to a door or cabinet, but the algorithm does not directly detects the handle. For it, they use a combination of several vision based techniques, to detect handles inside the ROI and its 3D positioning, with a complementary plane segmentation method, to differentiate door from the handle, which complicates the image processing prior to the grasp (Fig. 2.1). However, the use of a one-step CNN real-time method to detect the doors is a proposal that has inspired our project.

Since processing this kind of data involves the management of a lot of information, the main common limitation is the computation time that does not fulfill the requirements for real-time applications. Additionally, the assumption that only a single handle is present on the scene is made.

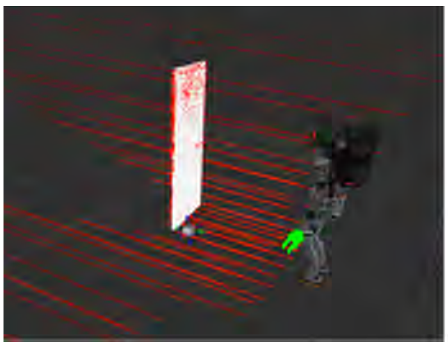
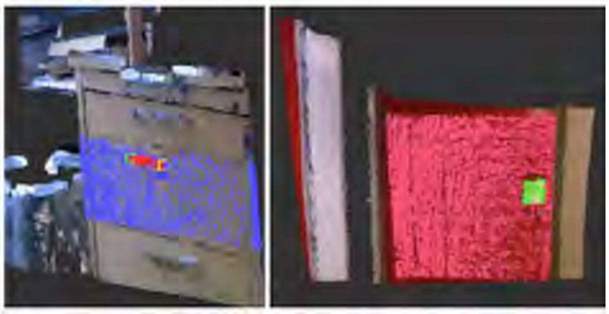



Figure 2.1: Approaches for door and handle detection proposed by Banerjee et al. [6], to the left, and by Llopart et al. [44]

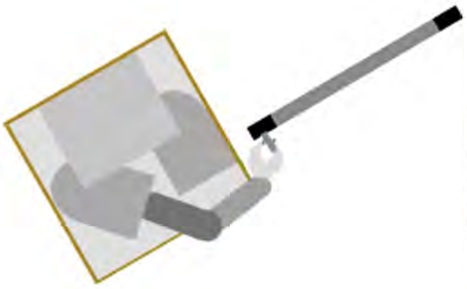
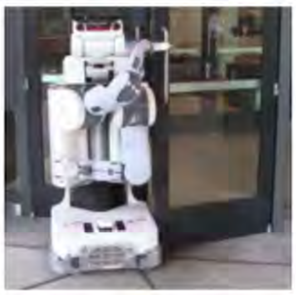
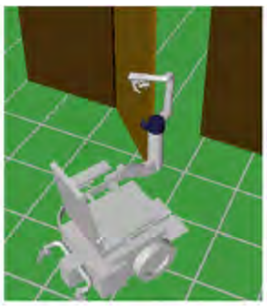




Figure 2.2: Model approaches for door manipulation proposed by Meeussen et al. [48], to the left, and by Diankov et al. [20], to the right

There is a variety of ways to open a door, which humans unconsciously choose and execute depending on the situation. Similarly, to adress the door manipulation problem with robotics systems different approaches have been proposed [22]. Most of these approaches assume substantial knowledge about the model and its parameters [50] or are model-free [31]. Although approaches relying on “a priori” parameters are constrained to a reduced set of doors, whereas model-free approaches release designers from providing any “a priori” model information, the knowledge about objects and their articulation properties supports state estimation, motion prediction, and planning.

Among the approaches that assume an implicit kinematic model of the articulated object, the following can be highlighted. Meeussen et al. [48] describe an integrated navigation system for mobile robots including vision and laser-based detection of doors and door handles that enables the robot to open doors successfully using a compliant arm (Fig. 2.2). Diankov et al. [20] formulate door and drawer operation as a kinematically constrained planning problem and propose to use caging grasps to enlarge the configuration space (Fig. 2.2). Wieland et al. [83] combine force and visual feedback to reduce the interaction forces when opening kitchen cabinets and drawers.

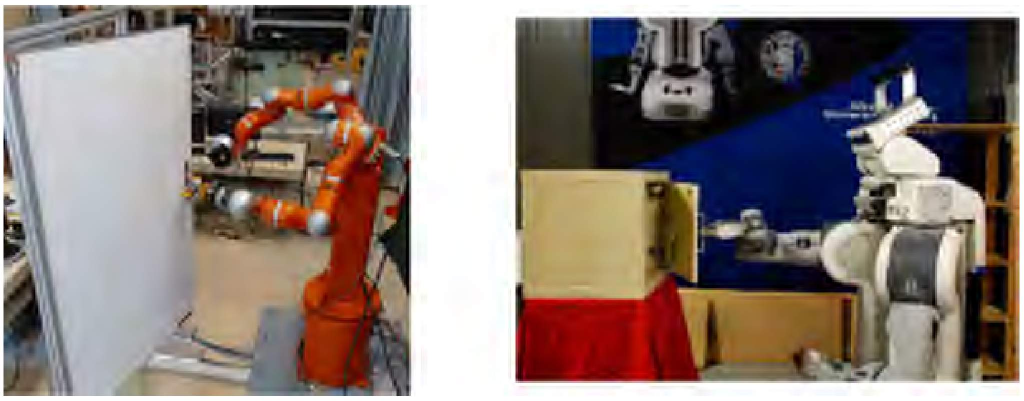


Figure 2.3: Probabilistic approaches for door manipulation proposed by Nemec et al. [51], to the left, and by Welschehold et al. [82], to the right

Among the model-free approach, a first solution could be those proposed by Lutscher et al. [45] and Karayiannidis et al. [31] approaches. They take advantage of the robot compliant behavior to accomplish the opening door task without estimating the direction of motion or the kinematics of the door. Lutscher et al. [45] propose to operate the unknown constrained mechanisms based on an impedance control method, which adjusts the guiding speed by impedance control to achieve two-dimensional plane operation. Karayiannidis et al. [31] propose a methodology that consists of a velocity controller which relies on force/torque measurements and estimation of the motion direction. These methods are suitable for any velocitiy controlled manipulator with a force/torque sensor at the end-effector. The force/torque control regulates the applied forces and torques within given constraints, while the velocitiy controller ensures that the end-effector of the robot moves with a task-related desired tangencial velocity. These methods, however, require a rich sensory feedback from the end-effector and advanced hardware capabilities of the robot in order to open a door. Another solution has been proposed by Banerjee et al. [6], with an approach that enables an ATLAS robot to open doors using motion planning with a optimiser that only involves kinematic constraints.

Finally, other works use probabilistic methods that are off-line and do not consider interaction force issues. Nemec et al. [51] use a reinforcement learning approach combined with intelligent control algorithm to open doors with an articulated robot (Fig. 2.3). Welschehold et al. [82] propose a probabilistic approach to learn joint robot base and gripper action models for door opening from observing demonstrations carried out by a human teacher (Fig. 2.3). The probabilistic approach proposed by Sturm et al. [76] could be considered the state-of-the-art, and has been adopted as a basic reference since it allows the robot to infere the kinematic model from observations of its motion. However, even if a framework for exploiting prior knowledge is proposed, it has not been analyzed extensively.

This work aims to overcome the current limitations and extend the state-of-the-art for the door opening task with a mobile manipulator robot.

3 Toyota Human Support Robot (HSR)

The Toyota Human Support Robot (HSR) is one of the Toyota Partner Robots (Figure 3.1) designed to support human activities and provide assistance to handicapped people for independent living at home. The HSR Robot has been designed to safely interact with people (Figure 3.2), and it is equipped with a folding arm. A force relaxation mechanism to prevent possible accidents from excessive external force applied on joint axes is used. Furthermore, obstacle avoidance helps the HSR to operate safely in a human-centric environment. It has a compact a lightweight body. To better accommodate a wide variety of households. It has a cylindrical telescopic body of 430mm in diameter, with a minimum height of 1005mm and weight of 37kg. Despite its compact footprint, the HSR can cover a relatively large workspace (Figure 3.3) [26].

The HSR has a differential drive type mobile base that consists of two drive wheels and three casters (Figure 3.4). The omni-wheels of the casters, located on the front and rear of the drive wheels, facilitate smooth changes in direction. Its articulated arm has seven degrees of freedom. The hand is a two-fingered gripper driven by a single motor and it is adopted for the end effector. The fingertips and the base of the fingers are designed to be compliant and the fingertip surface is covered by elastic material. This design enables the fingertips to conform to the shape of the objects while grasping, and prevent excessive force acting in the case of contact with environment and/or people. The rated fingertip force is 20N and the maximum grasping width is 140mm. The fingertip includes a vacuum pad to lift thin items such as cards or paper. The moving base is equipped with a laser range sensor used to measure the distance of obstacles when moving, a bumper sensor, used to detect contact with an obstacle, and a magnetic sensor able to detect magnetic tape and the like on the floor. The body trunk is equipped with an IMU (inertial measurement unit), which measures linear acceleration and rotational rate. The hand is equipped with a $6-$axis force sensor to detect forces on the wrist, and a camera. And, finally, the head has a RGB-D sensor, a stereo camera and a wide-angle camera [26].

The software consists of two layers, namely the real-time control layer and the intelligence layer. The real-time control layer executes time-critical processing such as motor control, while the intelligence layer processes computationally expensive logic or large data such as RGB images and point clouds. For the intelligence layer, Linux (Ubuntu) is used as the OS, and ROS (Robot Operating System) as the middleware [26].

The Human Support Robot (HSR) is the Toyota answer to the ever-increasing demand for long-term elderly care. Toyota is teaming up with a number of research institutes to found the HSR Developers' Community. This institute will put forth a cooperative effort to hasten the development and early practical adoption of the HSR. This robot represents the Toyota first step towards a society where robots exist in harmony with people, assisting their activities and mobility.

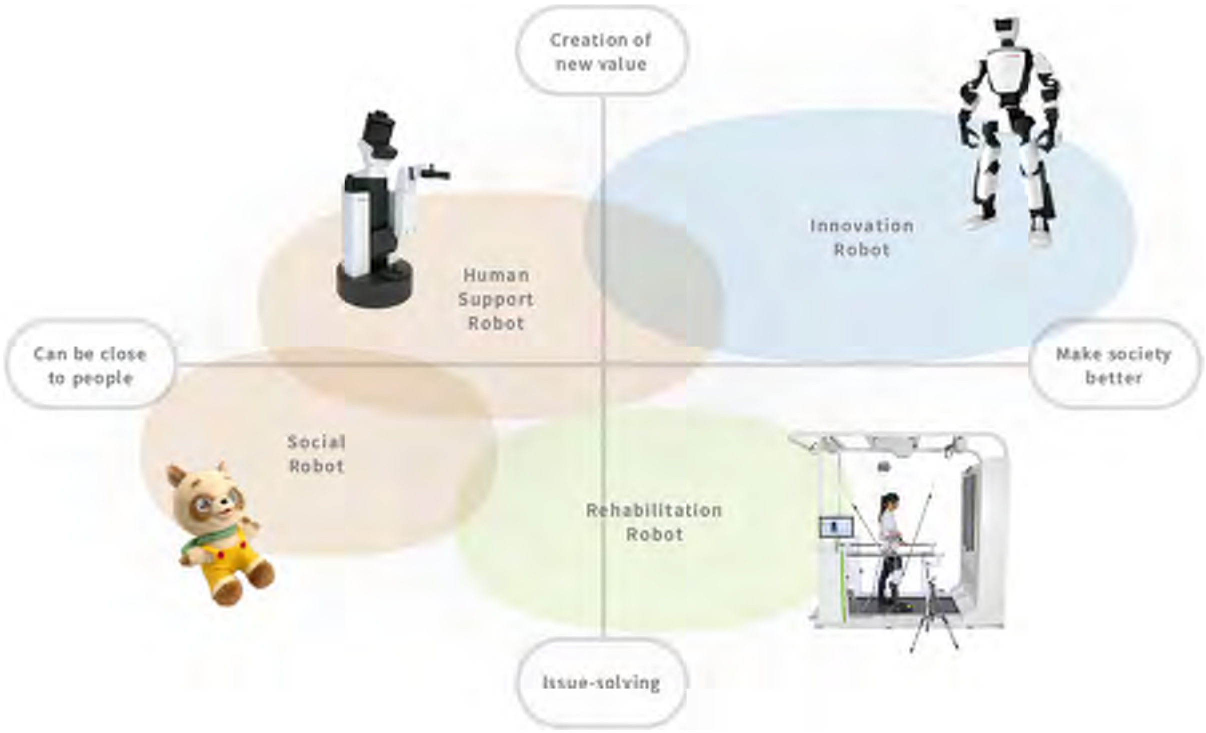


Figure 3.1: Toyota Partner Robots familiy

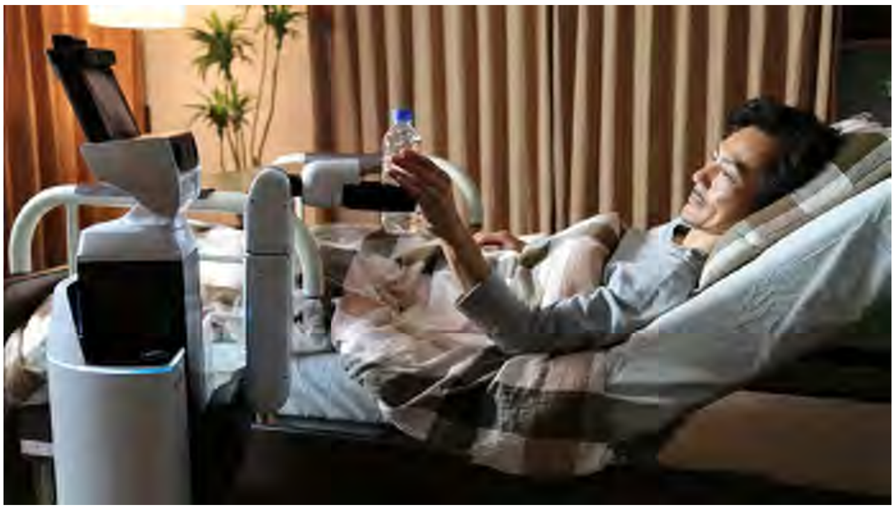


Figure 3.2: HSR provides assistance and life support

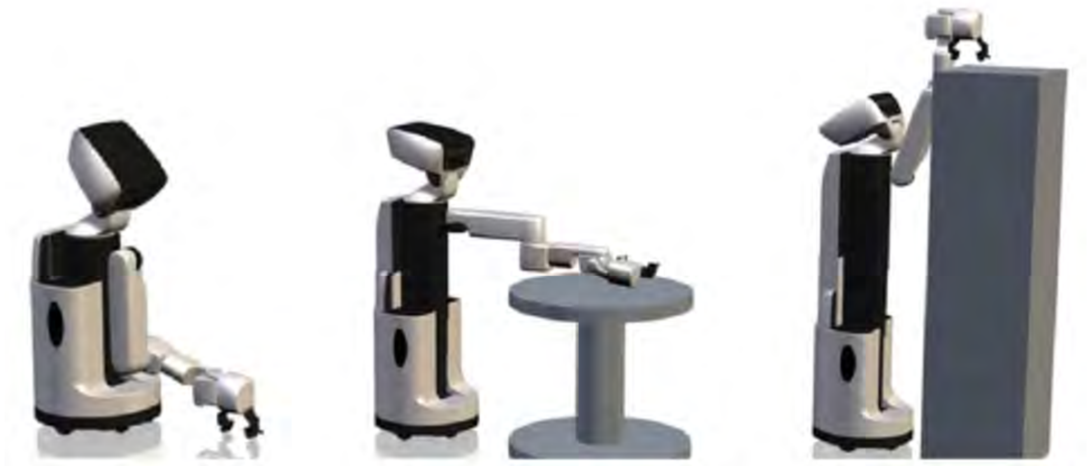


Figure 3.3: HSR telescopic body covers a large workspace

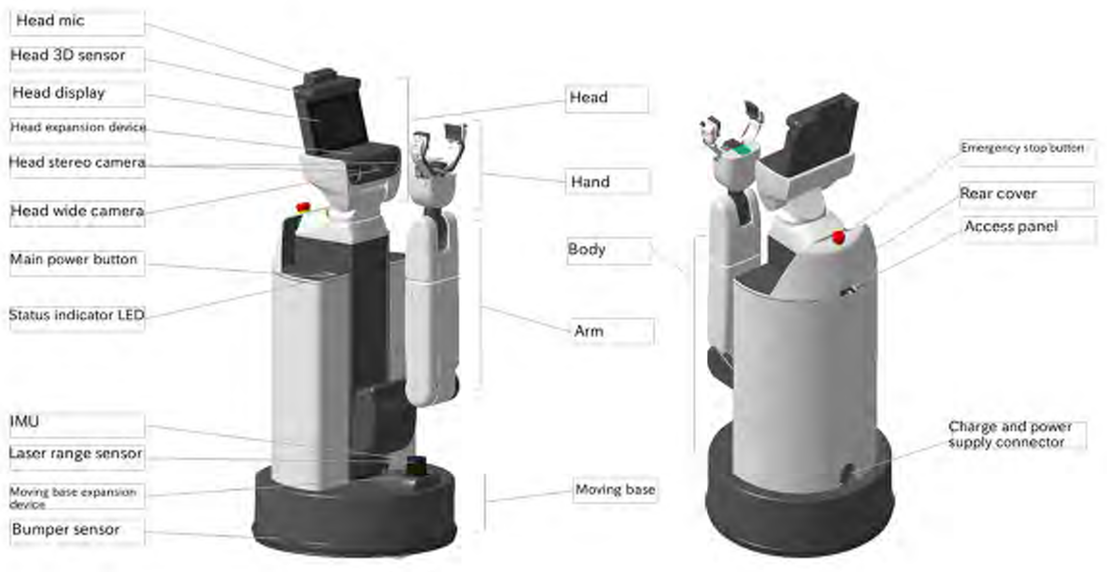


Figure 3.4: Configuration of the HSR

4 Perception

One of the major obstacles for reliable robotic autonomy is the problem of the extraction of useful information about the external environment from sensory readings or, in other words, robotic perception. Sensors can give robots the ability to see, touch and hear. Vision is used as a mean to act for the robot and interact with the world. Robotics systems rely on 3D perception in order to enable the holistic perception of their surroundings. The technology of 3D cameras has quickly evolved in recent years. Within robotics, these cameras open up the possibility of real-time robot interaction in human environments [2]. These cameras are also known as RGB-D cameras, because they are composed by a classical RGB camera and a depth sensor.

The Toyota Human Support Robot (HSR) incorporates an RGB-D camera in its head, in particular a Xtion PRO LIVE (Figure 4.1). This system performs triangulation between a light projector that throws a pattern with known structure into the scene, and a camera that views the scene and detects the pattern. By matching or correlating the pattern elements between the projector and the camera, and using the known relationship between them, it is possible to recover depth using standard methods. The camera/projector pair for depth recovery typically operate in the infrared to avoid the pattern being visible to the human eye [15].

In order to assist humans in performing ordinary tasks, a key issue is the interaction with objects. Object recognition and manipulation is something that seems very easy for a human and is hard for a robot. It is common to talk about a robot moving towards an object, but in reality the robot is only moving towards a pose at which it expects the object to be. This is a subtle but deep distinction. Thus, within a manipulating robot context, object recognition can be seen as a 6D-pose estimation problem.

To adress the problem of opening a door, the robot should be able to reach the handle and grasp it correctly. Exploiting the RGB-D data is essential to perceive the fundamental 3D structure necessary for its manipulation. In this chapter, an approach to estimate the key 3D geometric features of doors and handles combining computer vision object detection and 3D image processing algorithms is proposed. In section 4.1, a multiple object detection CNN that identifies different door clases and their corresponding handle is presented. Then, in section 4.2 an algorithm that estimates robustly the 3D geometric features of doors and handles in real time through the RGB-D sensor data is presented.

4.1 Handle and Door Detection Model

Humans recognize a multitude of objects in images with little effort, despite the fact that the image may vary somewhat depending on the point the view, the size and the scale or the position. This task is still a challenge for robotic systems. In order to

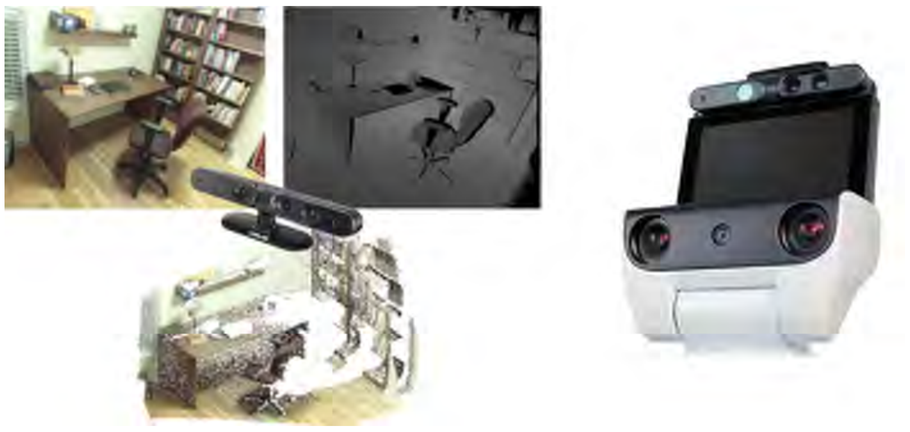


Figure 4.1: HSR RGB-D camera

be able to open doors, the robot must be endowed with the ability to recognize them and its corresponding handle. Since the robot may have to function in a wide variety of environments where doors and handles present a broad spectrum of colors, shapes, lighting conditions..., a robust detection is essential.

Object detection is one of the areas of computer vision that is maturing very rapidly thanks to Deep Learning and Convolutional Neural Networks (CNNs). Their capacity to from a set of examples, produce robust and reliable results makes them an excellent approach to adress the proposed problem.

4.1.1 Deep Learning and Convolutional Neural Networks

Until recently, most machine learning and signal processing techniques had exploited shallow-structured architectures. These architectures typically contain at most one or two layers of nonlinear feature transformations. Examples of the shallow architectures are Gaussian mixture models (GMMs), support vector machines (SVMs), logistic regression, kernel regression, and multilayer perceptrons (MLPs) with a single hidden layer including extreme learning machines (ELMs) [18]. Shallow architectures have been effective in solving many well-constrained problems, but their limited modeling and representational power can cause difficulties when dealing with more complicated real-world applications involving natural signals such as images [19].

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in object detection. This algorithm learns the objects features rather than being programmed with them. With multiple non-linear layers, i.e., a depth of 10 to 50 layers, a system can implement extremely intricate functions of its inputs that are simultaneously sensitive to minute details and insensitive to large irrelevant variations such as the background, pose, lighting and surrounding objects. Along with this advantage of such data-oriented classifiers, the drawnback is that an large amount of data is needed to achieve their performance [41].

The most common form of learning, deep or not, is supervised learning. It consists of learning a function that maps an input to an output based on example input-output pairs. The first step is, then, to collect a large data set of images. Later, the training process can start. During training, using the images collected, an objective function that measures the error (or distance) between the output scores and the desired pattern of scores is computed. Thise error is reduced iteratively adjusting certain parameters, often called weights, that define the input-output function. In a typical deep-learning system, there may be hundreds of millions of these adjustable weights. To adjust the weigth vector in each iteration, the gradient-descent method is the standard technique. When the error is small enough the model is trained, thus, it will be able to infere the desired output from new data.

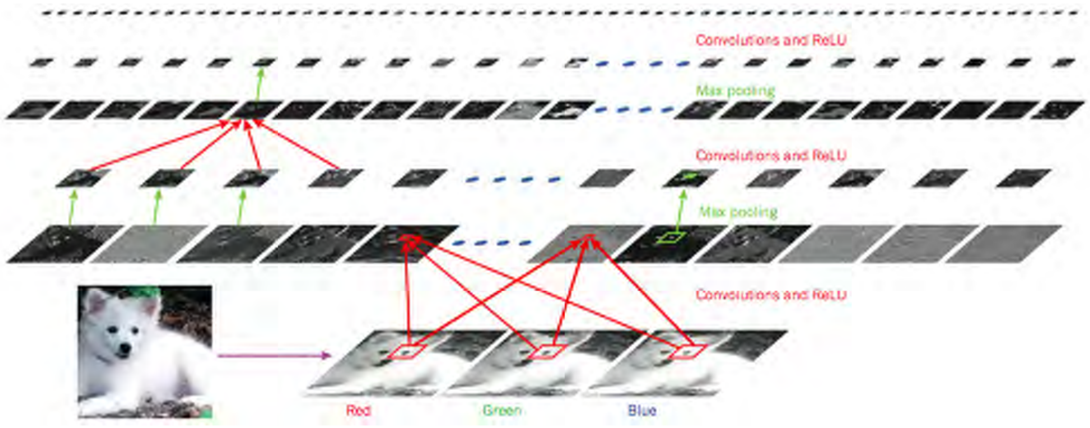


Figure 4.2: Architecture of a typical CNN [41]

Among the deep learning methods, the most representative model is Convolutional Neural Networks (CNNs). This group of methods is today the most capable, and they are able to handle many classes of object simultaneously and accurately classify them. There are four key ideas behind CNNs that take advantage of the properties of natural signals: local connections, shared weights, pooling and the use of many layers [41]. The architecture of a typical CNN is structured as a series of stages (Figure 4.2). The first few stages are composed of two types of layers: convolutional layers and pooling layers. Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of weights called a filter bank. The result of this local weighted sum is then passed through a non-linear function such as a ReLU. All units in a feature map share the same filter bank. The reason for this architecture is twofold. First, in array data such as images, local groups of values are often highly correlated, forming distinctive local motifs that are easily detected. Second, if a motif can appear in one part of the image, it could appear anywhere, hence the idea of units at different locations sharing the same weights and detecting the same pattern in different parts of the array. Mathematically, the filtering operation is a discrete convolution, hence the name.

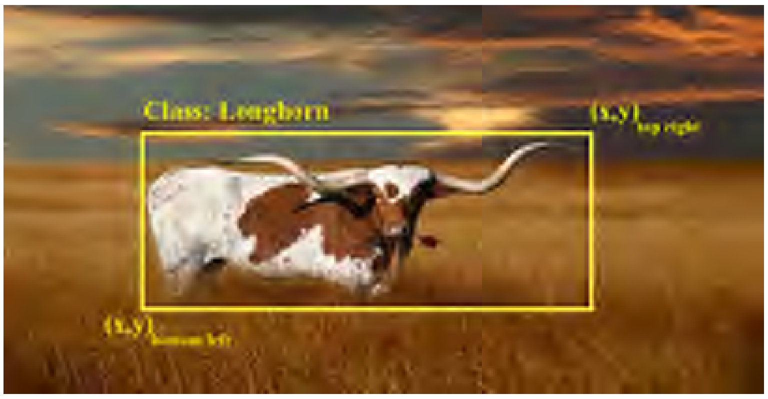


Figure 4.3: Longhorn bounding box

Although the role of the convolutional layer is to detect local conjunctions of features from the previous layer, the role of the pooling layer is to merge semantically similar features into one. Because the relative positions of the features forming a motif can vary somewhat, reliably detecting the motif can be done by coarse-graining the position of each feature. A typical pooling unit computes the maximum of a local patch of units in one feature map. Two or three stages of convolution, non-linearity and pooling are stacked, followed by more convolutional and fully-connected layers. The final stage is composed of fully connected layers which outputs an N dimensional vector where N is the number of classes that the network has to choose from by looking at the output of the previous determining which features most correlate to a particular class. Deep neural networks exploit the property that many natural signals are compositional hierarchies, in which higher-level features are obtained by composing lower-level ones. In images, local combinations of edges form motifs, motifs assemble into parts, and parts form objects. CNNs have been applied with great success to the detection, segmentation and recognition of objects and regions in images.

Object detection is the task of simultaneously classifying (what) and localizing (where) multiple object instances in an image. Given an image, a detector will output the coordinates of the location of an object with respect to the image. In computer vision, the usual way to localize an object in an image is to represent its location with bounding boxes, as shown in Figure 4.3. They can be defined either by the coordinates of two opposite corners, or the center or the widht and height.

The CNNs for generic object detection methods can be divided into two categories [86]:

- Region proposal: They follow a traditional object detection pipeline, generating region proposals at first and then classifying each proposal into different object categories. They are composed of several correlated stages, including region proposal generation, feature extraction with CNN, classification and bounding box regression, which are usually trained separately. As a result, the time spent in handling different components becomes the bottleneck in real-time applications.

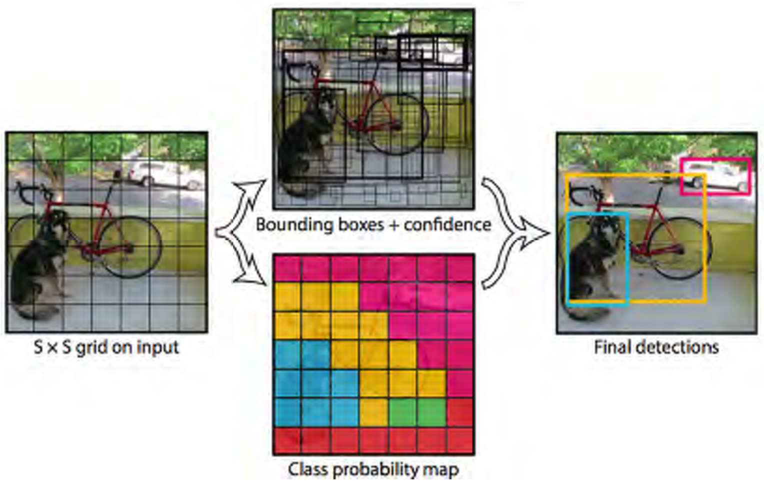


Figure 4.4: YOLO model detection as a regression problem [61]

- One-step: They regard object detection as a regression or classification problem, adopting a unified framework to achieve final results directly. They are based on global regression/classification, mapping straightly from image pixels to bounding box coordinates and class probabilities, reducing time expense. In short, regression based models can usually be processed in real-time at the cost of a drop in accuracy compared with region proposal based models.

The capacity of the regression based methods to process images in real time makes them very suitable for their integration into the vision systems used in robotics. In this work, You Only Look Once (YOLO), an open source and a state-of-the-art object recognition CNN for robot vision systems [61], will be used.

4.1.2 Object Detection: You Only Look Once (YOLO)

Redmon et al. [61] proposed a novel open source framework called YOLO, which unifies the separate components of object detection into a single neural network. This network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means this network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and real-time speeds while maintaining high average precision. YOLO divides the input image into an $S \times S$ grid and each grid cell is responsible for predicting the object centered in that grid cell. Each grid cell predicts B bounding boxes and their corresponding confidence scores. These confidence scores reflect how likely the box contains and object, and how accurate is the boundary box (Figure 4.4).

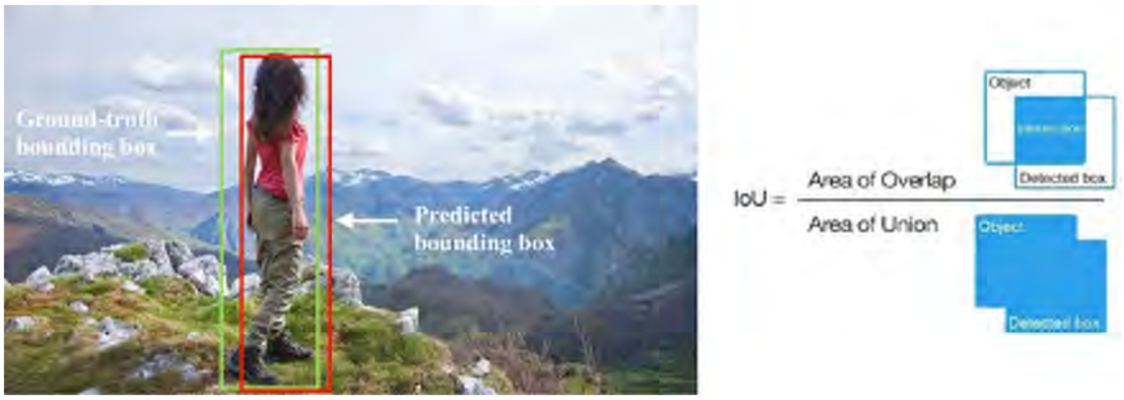


Figure 4.5: Intersection over Union (IOU) evaluation metric to measure the accuracy of an object detector

Formally, confidence scores are defined $Pr(\text{Object}) \times IOU_{pred}^{truth}$, where $Pr\,(\text{Object}) \geq 0$ corresponds to the likelihood that there exists an object, and IOU_{pred}^{truth} are the confidence of the bounding box prediction. IOU denotes the Intersection over Union metric, that is, the ratio of overlap and union areas. It is an evaluation metric used to measure the accuracy of an object detector on a particular dataset (Figure 4.5).

At test time, class-specific confidence scores for each box are achieved by multiplying the individual box confidence predictions with the conditional class probabilities as follows:

$$Pr\,(\text{Object}) \times IOU_{pred}^{truth} \times Pr\,(\text{Class}_\text{i} \mid \text{Object}) = Pr\,(\text{Class}_\text{i}) \times IOU_{pred}^{truth}$$

where the existing probability of class-specific objects in the box and the fitness between the predictor box and the object are both taken into consideration.

At training, the following loss function is optimized:

$$\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \urcorner_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \urcorner_{ij}^{obj} \left[\left(\sqrt{w}_i - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h}_i - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \urcorner_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \urcorner_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \urcorner_{ij}^{obj} \sum_{c \in classes} \left[p_i\,(c) - \hat{p}_i\,(c) \right]^2 ,
\end{aligned} \tag{4.1}$$

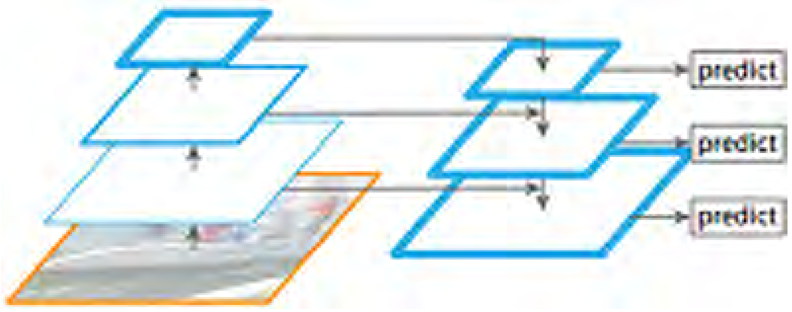


Figure 4.6: Feature Pyramid Network

where, in a certain cell i: (x_i, y_i) denote the center of the box relative to the bounds of the grid cell; (ω_i, h_i) are the normalized width and height relative to the image size; C_i represents confidence scores; $\urcorner_i^{obj}$ denotes if the object appears in the cell; $\urcorner_{ij}^{obj}$ denotes that the jth bounding box predictor in the cell is "responsible" for that prediction; $\urcorner_i^{noobj}$ denotes if the object does not appear in the cell; $\urcorner_{ij}^{noobj}$ denotes that the jth bounding box predictor in the cell is "responsible" for that prediction; λ_{coord} is a parameter to increase the loss from bounding box coordinate predictions; and λ_{noobj} is a parameter to decrease the loss from confidence predictions for boxes that do not contain objects. YOLO predicts multiple bounding boxes per grid cell. At training time only one bounding box predictor is responsible for each object. Thus, we assign one predictor to be "responsible" for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors.

Since the first version came out, Redmon et al. [62] have developed improved versions of this family of algorithms for object recognition. The last version is YOLOv3 [63]. YOLOv3 has a fully convolutional architecture. No fully-connected layer is used. This structure makes it possible to deal with images with any sizes. Also, no pooling layers are used. Instead, a convolutional layer with stride 2 is used to downsample the feature map, passing size-invariant feature forwardly. This helps to prevent loss of low-level features often attributed to pooling. Additionally, YOLO v3 makes predictions at three scales. They are designed for detecting objects with various sizes. Then features are extracted from each scale by using a method similar to that of feature pyramid networks (FPN) (Figure 4.6). In these type of networks, the prediction is done on feature maps at different depths, but further features are utilized by upsampling the feature map and merging it with current feature map.

FPN composes of a bottom-up and a top-down pathway. The bottom-up pathway is the usual convolutional network for feature extraction while the top-down pathway constructs higher resolution layers from a semantic rich layer. With this technique, the network is more capable to capture the object information, both low-level and high-level.

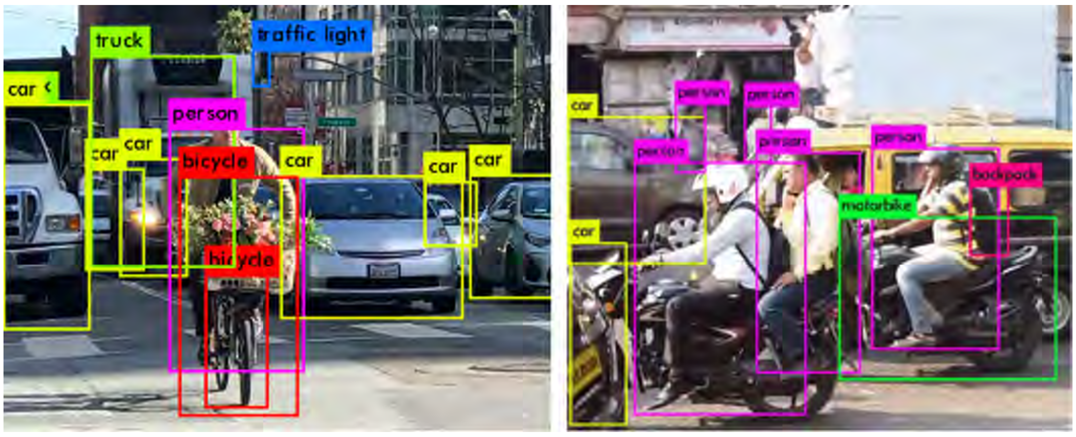


Figure 4.7: Examples of object detection with YOLOv3

All these features have made YOLO the most popular object detection CNN, since it is able to achieve processing rates of 30 fps in a Pascal Titan X GPU with great accuracy. Some examples are shown in Figure 4.7.

4.1.3 Detected Classes

The most simple classification could be “door” and “handle”. Where “door” refers to a general definition that includes room doors, drawers, lockers... and “handle” refers to the actuation mechanism that operates the door. However, to make the proposed approach as versatile and extendable to other applications as possible, the class “door” was splitted in three classes. In this way, the object detection model will provide additional information. Thus, the detected classes of the proposed model are (some representative examples of what objects includes each class are shown in Figure 4.8): (a) Door, (b) Cabinet door, (c) Refrigerator door, and (d) Handle.

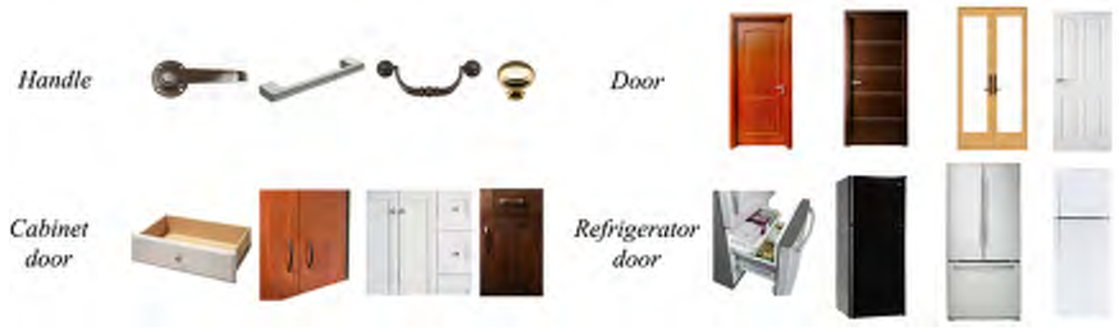


Figure 4.8: Detected classes

4.1.4 Training Data

Deep-learning based model require labelled data to adjust their parameters and provide the desired output. In the context of object detection these data refers to a set of images with their corresponding bounding boxes.

There is not a general rule to determine the minimum number of images to achieve a satisfactory performance in object detection. However, there is a general consensus that usually "the more, the better". It should be highlighted that the quality of the pictures is important, not just the quantity (Figure 4.9).

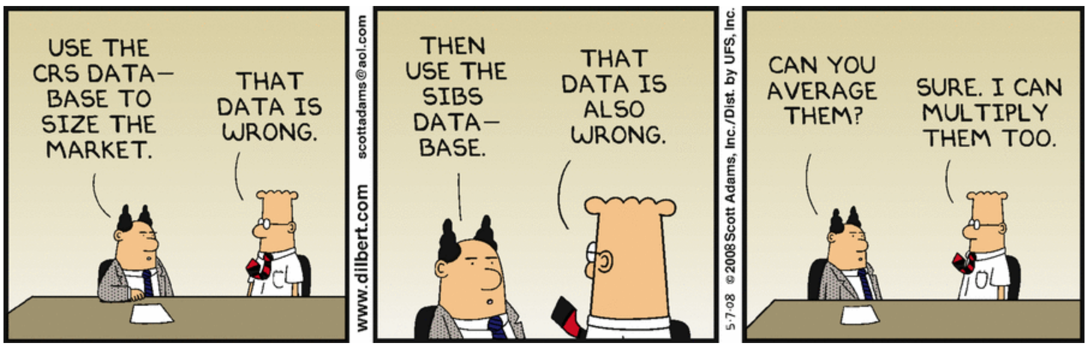

USE THE CRS DATA-BASE TO SIZE THE MARKET.
THAT DATA IS WRONG.
scottadams@aol.com
www.dilbert.com
THEN USE THE SIBS DATA-BASE.
THAT DATA IS ALSO WRONG.
5-7-08 © 2008 Scott Adams, Inc./Dist. by UFS, Inc.
CAN YOU AVERAGE THEM?
SURE. I CAN MULTIPLY THEM TOO.

Figure 4.9: Gilbert shows that the quality of the data is very important

It is very important that the training images are as close as possible to the inputs that the model will see when it is deployed. Gathering the correct training data is key for our CNN performance. However, well-annotated data can be both expensive and time consuming to acquire. Preparing large training sets for object recognition has already become one of the main bottlenecks for such emerging applications as mobile robotics and object recognition on the web. The two possible procedures to collect the data are:

- Download it from a fully-labeled image dataset: Well-annotated data can be both expensive and time consuming to acquire. There are several open-source datasets that can be used for bechmarking object detection CNNs. The most popular ones are: ImageNet, Pascal VOC and MS COCO.

- Make your own training data from scratch: This should be done when the class that is desired to be identified by the model is not available in the already-existing image datasets.

The classes presented in the previous section are not identified among the existing data sets, they are usually split in many other different categories. Furthermore, the usual definition of handle only includes the door handles, and for the project a more extended definition is needed. For these reasons, in order to identify the desired classes, the data set had to be made from scratch.

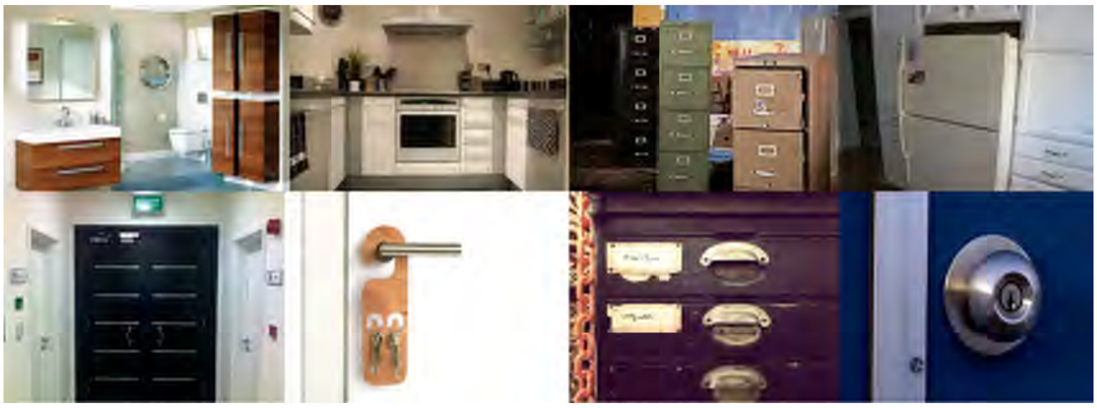


Figure 4.10: Object particular classes in Open Images Dataset V4

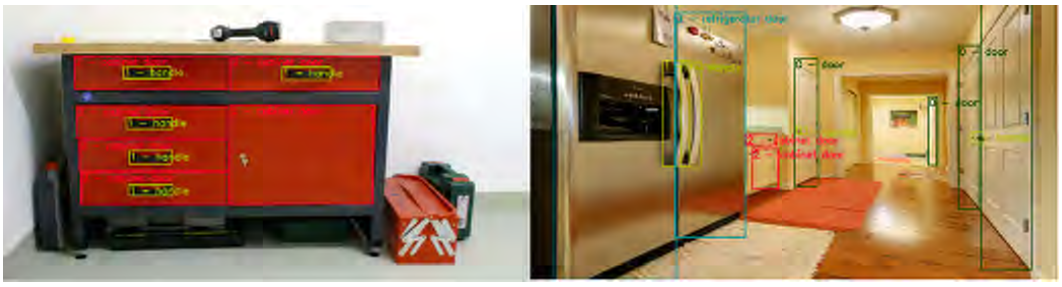


Figure 4.11: Examples of labeled images, available at [4] GitHub repository

The first step is to gather the images that will be used for further annotation. Instead of downloading images one by one with a common search engine, a simpler process is to download certain classes from an image dataset without the annotations and label them manually. In this project the Open Images Dataset [55] was used. Some examples of the images used for the training data are shown in Figure 4.10. Open Images is a dataset of $\sim$ 9M images that have been annotated with image-level labels and object bounding boxes. The training set of V4 contains 14.6M bounding boxes for 600 object classes on 1.74M images, making it the largest existing dataset with object location annotations. The images are very diverse and often contain complex scenes with several objects. The dataset is a product of a collaboration between Google, CMU and Cornell universities. Once the images have been gathered and selected, the next step is to label the data. Manual image annotation is the process of manually defining regions in an image and creating a textual description of those regions. There exists different tools for this task such as LabelMe (CSAIL, MIT), Labelbox, or RectLabel. Given a target class, a perfect box is the smallest possible box that contains all visible parts of the object [36]. Taking this premise into account a total of approximately 1200 images were manually labelled for the training dataset (Figure 4.11). This dataset is fully available at [4] GitHub repository.

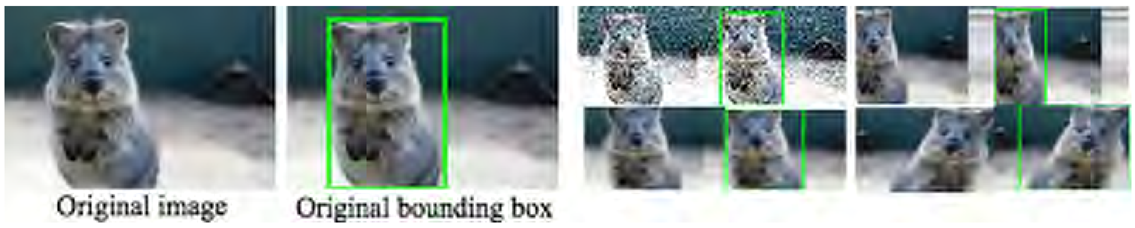


Figure 4.12: Data augmentation applied to the training data

Separating data into training and testing sets is an important part of evaluating the model. After the model has been processed by using the training set, the model is tested by making predictions against the test set. Thus, the labelled data was splitted randomly in 200 images for testing and 1000 images for training. Finally, to improve the training process, Imgaug library was used to apply data augmentation techniques to the training data. That is, generating synthetic training examples by altering some of the original images from the data set, while using the original class label label (Figure 4.12). Performing data augmentation for learning deep neural networks is known to be important for training visual recognition systems by improving generalization [77]. Once the data is gathered, the training process can start.

4.1.5 Training the Model

A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow the algorithm to correctly determine the class labels for unseen instances. Deep learning needs considerable hardware to run efficiently. The computationally intensive part of neural network is made up of multiple matrix multiplications in the training process. It can be made faster by computing all the operations at the same time. Thus, a GPU (graphical processing unit) is highly recommendable for training a CNN. Their highly parallel structure makes them more efficient than general-purpose CPUs for algorithms that process large blocks of data in parallel. An intuitive example to understand the difference between a CPU and a GPU could be the following: To transfer goods from one place to another a Ferrari or a freight truck could be either used. The Ferrari would be extremely fast but the amount of goods that can be carried is small, and usage of fuel would be very high. A freight truck would be slow and would take a lot of time to transfer goods. But the amount of goods it can carry is larger in comparison to the Ferrari. Also, it is more fuel efficient. The Ferrari is the CPU and the freight truck is the GPU.

To train the model proposed in this project an NVIDIA Geforce GTX 1080 was used. For starting the training process, pre-trained weights on large datasets were used, in particular DarkNet53 trained on ImageNet. By using these weights this previous learning is applied to the proposed problem statement. This is known as transfer learning.

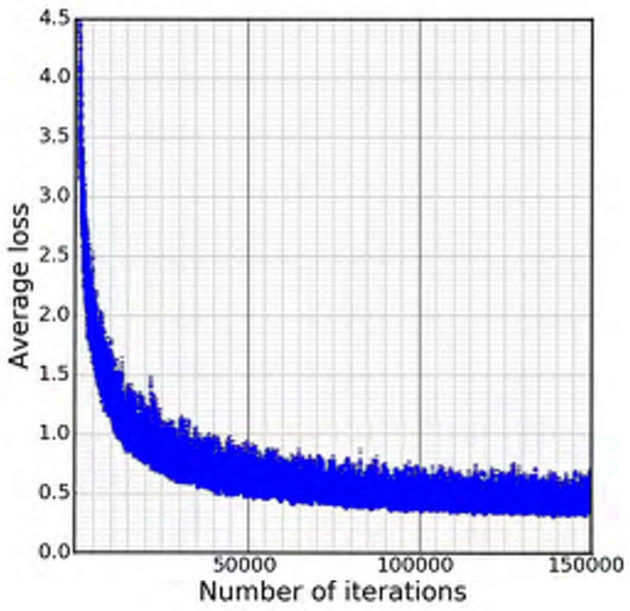


Figure 4.13: Learning curve

In this way, the time to train the dense layer of the network is comparatively neglibible and is done with greater accuracy. A total of approximattely 150000 iterations were done during the training process (Figure 4.13). It was stopped when the average loss stabilized around 0.5, taking the process one week.

4.1.6 Testing the Model

The performance of the proposed CNN will be judged using the mAP (mean average precision) criterium defined in the PASCAL VOC 2012 competition. mAP is the metric to measure the accuracy of object detectors. It is a number from 0 to 100, and higher values are typically better. Briefly, the mAP computation is done in the following way:

- Each bounding box will have a score associated (likelihood of the box containing an object).
- Based on the predictions, a precision-recall curve (PR curve) is computed for each class by varying the score threshold. The average precision (AP) is the area under the PR curve.
- First the AP is computed for each class, and then averaged over the different classes.

Being the end result the mAP.

The first step is, then, to make the PR curve, for which precision and recall should be defined:

- Precision (in statistics also known as positive predictive value or PPV) measures how accurate are the predictions, i.e., the proportion of positive predictions that are correct.
- Recall (in statistics also known as sensitivity) measures the proportion of actual positives that are correctly identified as such.

Mathematically, they are defined as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \quad ; \quad \text{Recall} = \frac{TP}{TP + FN}$$

where: TP=True Positive, TN=True Negative, FP=False Positive, and FN=False Negative.

In the terminology true/false and positive/negative, true or false refers to the assigned classification being correct or incorrect, while positive or negative refers to assignment to the positive or the negative category, thus positive corresponds an object has been detected and negative the opposite. For the PASCAL VOC challenge, a prediction is true if $IOU > 0.5$, false otherwise. To compute the precision-recall curve (PR) for each class the test images that were splitted from the training data are used. The ground-truth will correspond therefore to the manually set bounding-boxes.

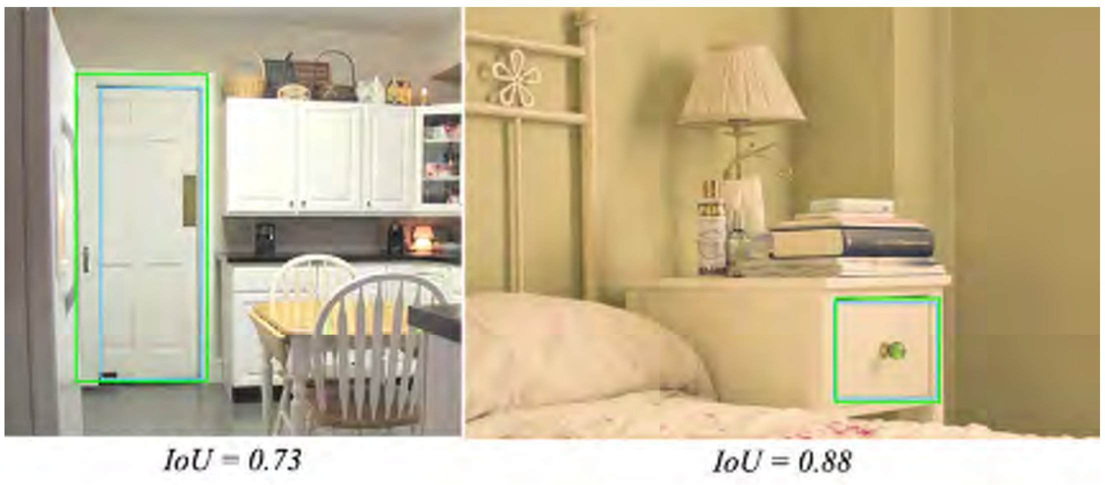


Figure 4.14: IOU evaluation

The CNN predicted objects for the whole set are sorted by decreasing confidence and are assigned to ground-truth objects (Figure 4.14). Then, from most confident to less, the IOU is computed. From each new detection, the accumulated TP, FP, TN and FN are updated. Thus, a new point of the PR curve is obtained. A linear interpolation is used to create the curve.

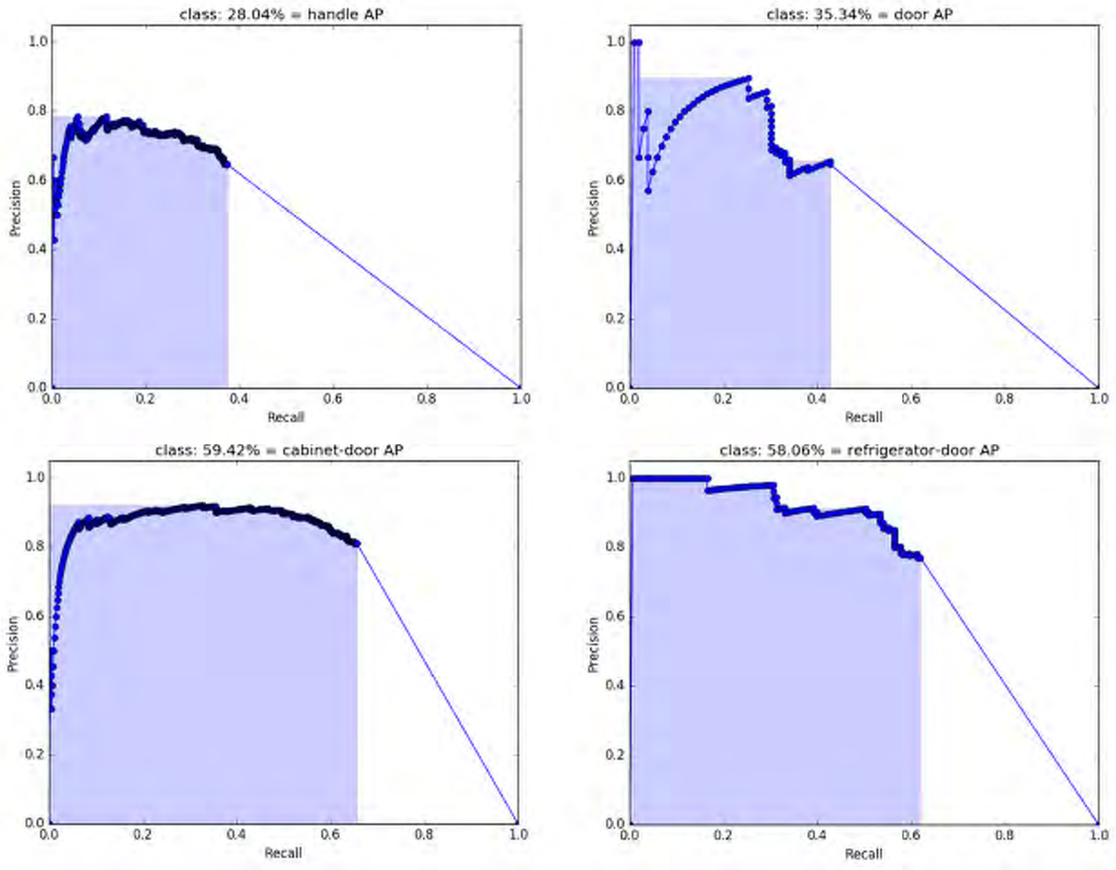


Figure 4.15: PR curve per classes

The AP is calculated for each class as the area under the curve shown in light blue (Figure 4.15), which is an approximate precision/recall curve with precision monotonically decreasing, by setting the precision for recall r to the maximum precision obtained for any recall $r^{'} > r$, by numerical integration. No approximation is involved since the curve is piecewise constant. Finally, the mAP is directly calculated as the average AP. The resulting mAP is 45% (Figure 4.16). The proposed model has a slightly inferior value for mAP compared to the other examples. This could be explained by the inferior size of the data set. However, in terms of performance, very satisfactory results are obtained. Finally, the proposed object detector is tested in real experiments. The network runs at a speed of 20 fps using an NVIDIA GeForce GTX 1080.

To give an idea of what does this value mean in terms of performance we have compared it with different YOLO versions in standard datasets (Table 4.1).

The results are shown in Figure 4.17. As it can be observed the detections are very accurate. Thus, it is concluded that the approach is really efficient for detecting handles and different classes of doors in real-time.

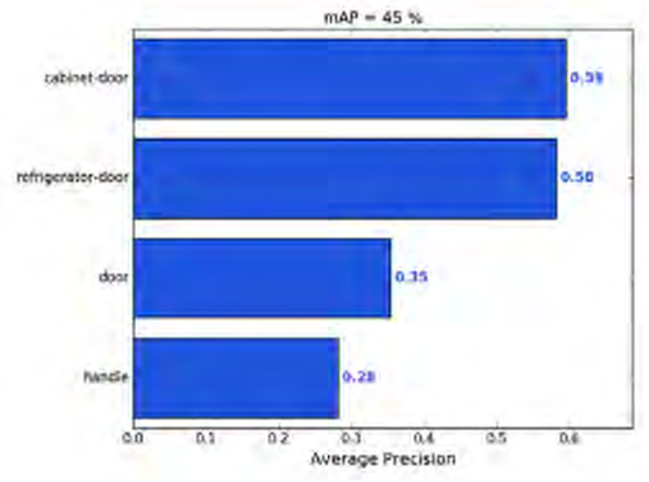


Figure 4.16: *AP* results per classes

	mAP
YOLOv3 on COCO dataset	55%
YOLOv2 on COCO dataset	48%
YOLOv1 on VOC 2012 dataset	58%
Proposed model	45%

Table 4.1: Results comparison

At this point, a robust detection is achieved. However, as it was explained in the previous sections, in order to grasp the handle and operate the door, the robot needs an estimation of the 3D position and orientation of the object. To adress this problem, the depth information must be exploited.

4.2 Computation of the 3D Geometric Features of Doors and Handles

There has been a lot of activity in the development of both hardware and software in 3D imaging systems which is having a huge impact in the capabilities of robotics. The world is 3D and robots need this perception to interact with the enviroment. Using an RGB-D sensor, the data is in form of sets of 3-dimensional points which are often referred to as a point cloud (Figure 4.18). However, a lot of information is contained and extracting the valuable features for manipulating doors requires a deep understanding of its nature and the application of complex algorithms.

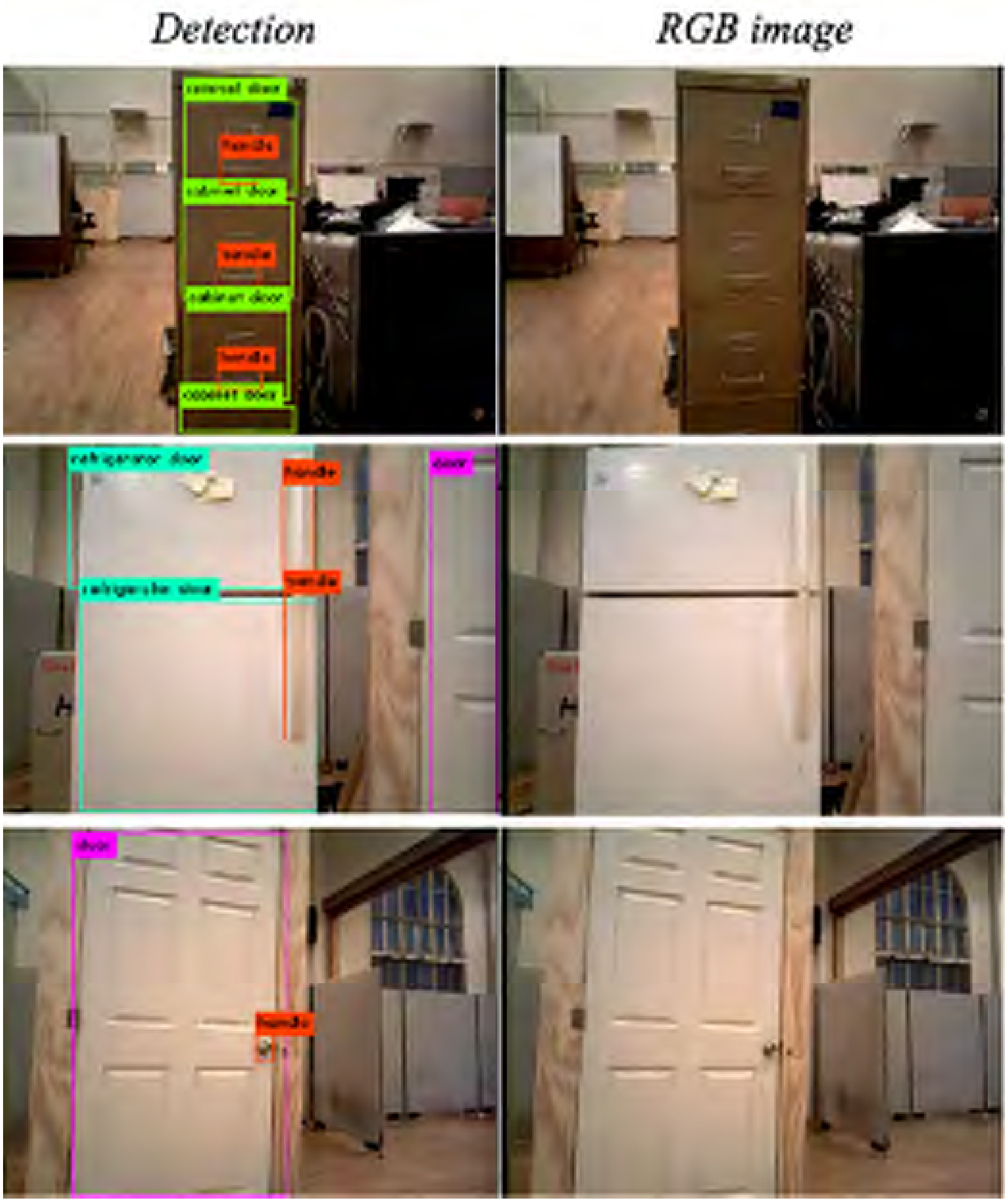


Figure 4.17: Results obtained with the proposed handle and door detection model

Figure 4.18: HSR perceives the world as a point cloud

The Point Cloud Library (PCL) is a large scale, open project for point cloud processing [67]. The PCL framework contains numerous state-of-the-art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. These algorithms can be used, for example, to filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene, extract keypoints and compute descriptors to recognize objects in the world of their geometric appearance. It can also create surfaces from point clouds and visualize them. In this regard, the work of R.B. Rusu [66] has been used as the main reference to address the problems raised in this project.

4.2.1 Point Cloud Filters

Filtering is an area of intensive research and the crucial step of the processing pipeline for a wide range of applications. Raw point clouds captured by 3D sensors contain a large amount of point samples, but only a small fraction of them are of interest. Furthermore, they are unavoidably contaminated with noise.

For these reasons, in order to manage point cloud data for real-time applications and to extract the desired geometric features precisely, it needs to be filtered adequately. The following filters of the PCL can be useful to adress this problems:

- ExtractIndices filter: Point Cloud data can be seen as an array. The extract indices filter performs a simple filtering along a specified set of indexes. It iterates through the entire point cloud once, performing two operations. First, it removes non-finite points. Second, it removes any points that are not contained in the input set of indices.

- StatisticalOutlierRemoval filter: Laser scans typically generate point cloud datasets of varying point densities. Additionally, measurement errors lead to sparse outliers which corrupt the results even more. This complicates the estimation of local point cloud characteristics such as surface normals or curvature changes, leading to erroneous values, which in turn might cause point cloud registration failures. Some of these irregularities can be solved performing a statistical analysis on each point neighborhood, and trimming those which do not meet a certain criteria.

  Statistical analysis is carried out on the discrete points, calculating average distance from every point to all its neighboring points and filtering the outliers outside the reference ranges of average distance from the data set. By assuming that the resulted distribution is Gaussian with a mean and a standard deviation, all points whose mean distances are outside an interval defined by the global distances mean and standard deviation can be considered outliers and trimmed from the dataset.

  The algorithm is [28, 66]:

  - Set k, an integer representing the number of closest point around point p_i.
  - For every point p_i in the entire 3D point cloud $\mathcal{P}$: (a) The location of their k nearest neighbors is found; and (b) the average distance $\bar{d_i}$ from point p_i to its k nearest neighbors is computed.
  - The mean μ_d of the average distances $\bar{d_i}$ is computed:

    $$\mu_d = \frac{1}{n}\sum_{i=1}^{n}\bar{d_i}$$

  - The standard deviation σ_d of the average distances $\bar{d_i}$ is computed:

    $$\sigma_d^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\bar{d_i} - \mu_d)^2$$

  - Assuming the distribution is Gaussian, a threshold T is computed:

    $$T = \mu_d + \alpha \cdot \sigma_d$$

    (where α is a standard deviation multiplier, set by the user as density restrictiveness factor).
  - Those points for which its average distance, $\bar{d_i}$, to its k neighbors is $\bar{d_i} > T$ are trimmed for the point cloud.
  - The remaining point cloud $\mathcal{P}^*$ is simply estimated as follows [66]:

    $$\mathcal{P}^* = \left\{p_j \in \mathcal{P} \mid (\mu_d - \alpha \cdot \sigma_d) \leq \bar{d_j} \leq (\mu_d + \alpha \cdot \sigma_d)\right\}$$

  The implementation therefore requires two steps over the whole set of points in the cloud:

  - First step to compute the mean and standard deviation of the average distances from each points in the cloud to their $k-$neighbors.
  - Second step to eliminate all the points in the cloud whose average distance (computed in the first step) is greater than the threshold.

  It is convenient to point out that: (a) while the algorithm will remove points whose average distance to its k nearest neighbors follows any statistical distribution, its parameters μ and σ are only meaningful for a normal (Gaussian) distribution; and (b) the algorithm assumes the capability to obtain the k nearest neighbors of any given point in the point cloud.

- VoxelGrid filter: The voxel grid filter allows to "prune" extra points from the cloud, down-sampling the pointcloud considerably, and reducing the number of points in a cloud using a voxelized grid approach. Unlike others sub-sampling filters, that only return a sub-set of points from the original point cloud, the voxel grid filter returns a point cloud with a smaller number of points but also maintains the shape characteristics of point cloud which best represent the input point cloud as a whole. This filter creates a 3D voxel grid (Figure 4.19) (think about a voxel grid as a set of tiny 3D boxes in space) over the input point cloud data. In each voxel, all the points present will be approximated (i.e., downsampled) with their centroid. Then, for each voxel A, the centroid is obtained as:

$$\bar{x} = \frac{1}{s} \sum_{(x,y,z)\in A} x \quad ; \quad \bar{y} = \frac{1}{s} \sum_{(x,y,z)\in A} y \quad ; \quad \bar{z} = \frac{1}{s} \sum_{(x,y,z)\in A} z$$

  where s is the total number of discrete points in voxel A. This approach is a slightly than approximating them with the center of the voxel, but it represents the underlying surface more accurately.

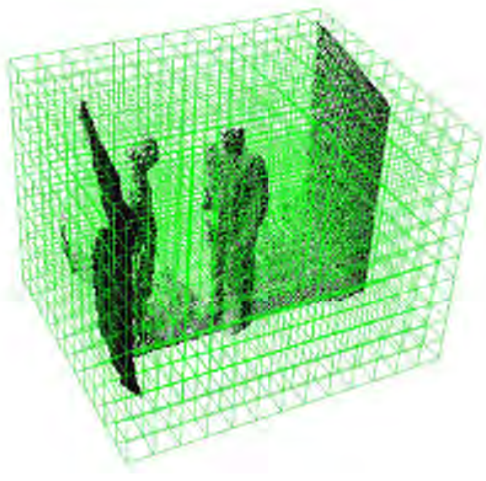


Figure 4.19: 3D Voxel Grid [81]

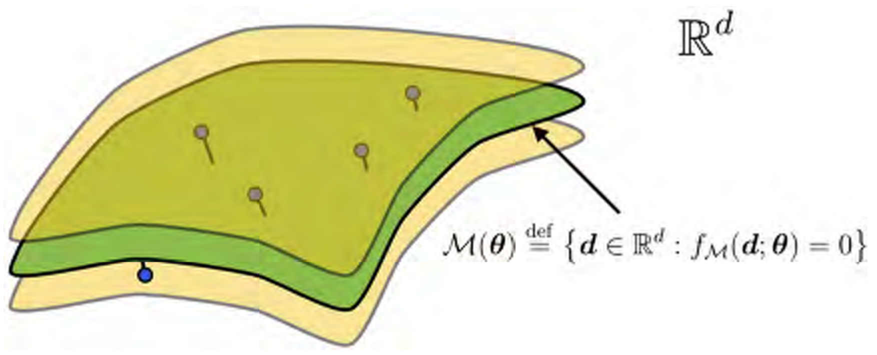

$\mathbb{R}^d$
$\mathcal{M}(\boldsymbol{\theta}) \overset{\text{def}}{=} \{\boldsymbol{d} \in \mathbb{R}^d : f_{\mathcal{M}}(\boldsymbol{d};\boldsymbol{\theta}) = 0\}$

Figure 4.20: The model space $\mathcal{M}$ as a green surface (the locus for which $f_{\mathcal{M}}(\mathbf{d}, \boldsymbol{\theta}) = 0$). The yellow surfaces represent the boundaries for a datum to be considered an inlier. The inliers (blue dots) lie in between the two yellow "crusts" [87]

4.2.2 RANdom SAmple Consensus (RANSAC)

The RANdom Sample Consensus (RANSAC) algorithm is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, where the outliers do not influence the values of the estimates. The estimation of the model parameters and the construction of the consensus samples (i.e. elements of the entire dataset are consistent with the model instantiated by the estimated model parameters) will develop as described bellow [87]:

The input dataset, of N elements, is $D = \{\mathbf{d}_1, ..., \mathbf{d}_N\}$. Let $\boldsymbol{\theta}(\{(\mathbf{d}_1, ..., \mathbf{d}_h)\})$ be the parameter vector estimated using the set of data $\{\mathbf{d}_1, ..., \mathbf{d}_h\}$, where $h \geq k$, and k is the cardinality of the minimal sample set (MSS). The model space $\mathcal{M}$ is defined as (see Figure 4.20):

$$\mathcal{M}(\boldsymbol{\theta}) = \{\mathbf{d} \in \mathbb{R}\colon \mathrm{f}_{\mathcal{M}}(\mathbf{d}; \boldsymbol{\theta}) = 0\}$$

where $\boldsymbol{\theta}$ is a parameter vector and $f_{\mathcal{M}}$ is a smooth function whose zero level set contains all the points that fit the model $\mathcal{M}$ instantiated with the parameter vector $\boldsymbol{\theta}$.

The error associated with the datum $\mathbf{d}$ with respect to the model space as the distance from $\mathbf{d}$ to $\mathcal{M}(\boldsymbol{\theta})$ is:

$$e_{\mathcal{M}}(\mathbf{d}, \boldsymbol{\theta}) = \min_{\mathbf{d}' \in \mathcal{M}(\boldsymbol{\theta})} dist(\mathbf{d}, \mathbf{d}')$$

where $dist(.,.)$ is an appropiate distance function.

Using this error metric, the Consensus Set (CS) is:

$$S(\boldsymbol{\theta}) = \{\mathbf{d} \in D : e_M(\mathbf{d}, \boldsymbol{\theta}) \leq \delta\}$$

where δ is a threshold that can either be inferred from the nature of the problem or, under certain hypotesis, estimated automatically.

RANSAC is a non-deterministic algorithm since it finishes when the probability of finding a better CS drops below a certain threshold. If ϵ is the probability of picking a sample that produces an outlier, then $(1-\epsilon)$ is the probability of picking at least one inlier. This means that the probability of picking a number s of inliers becomes $(1-\epsilon)^s$. For a number k of iterations, the probability of failure becomes $[1-(1-\epsilon^s)]^k$. Then, if p is the desired probability of success, the number of iterations should be:

$$1-p = [1-(1-\epsilon)^s]^k \;\Rightarrow\; k = \frac{\log(1-p)}{\log[1-(1-\epsilon)^s]} \tag{4.2}$$

In the original formulation of RANSAC, the ranking r of a consensus set was nothing but it cardinality:

$$r(CS) = \mid CS \mid$$

Thus, RANSAC can be seen as an optimization algorithm that minimizes the cost function defined as:

$$C_{\mathcal{M}}(D;\boldsymbol{\theta}) = \sum_{i=1}^{N} \rho\left[\mathbf{d}_i, M(\boldsymbol{\theta})\right]$$

where:

$$\rho\left[\mathbf{d}, M(\boldsymbol{\theta})\right] = \begin{cases} 0 & if \quad \mid e_M(\mathbf{d},\boldsymbol{\theta}) \mid \leq \delta \\ t & if \quad otherwise \end{cases}$$

and where δ is a user give threshold for considering inliers.

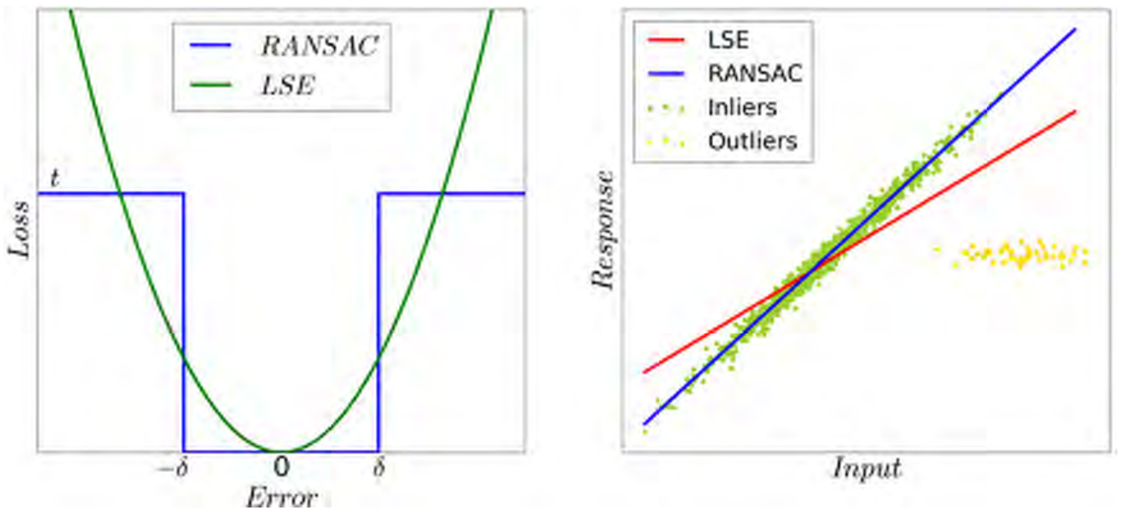


Figure 4.21: Loss functions for Least Squares and RANSAC methods. While Least Squares method (green) highly penalizes gross errors, the “top-hat” cost function of RANSAC (blue) only counts the number of inliers [14]

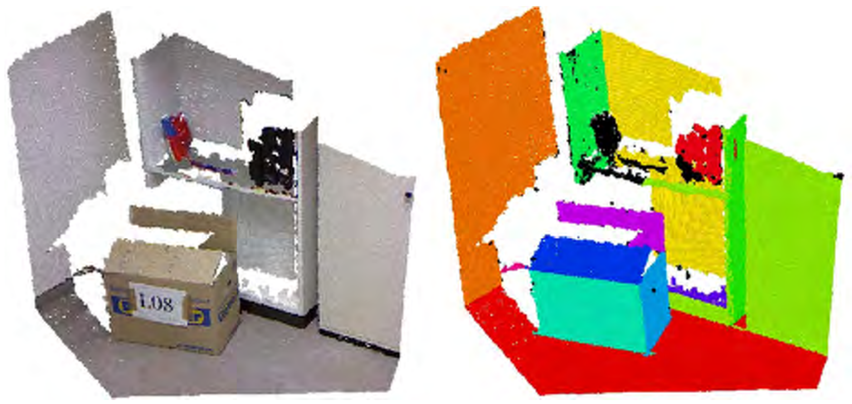


Figure 4.22: Planar segmentation of a scene using RANSAC [54]

The thresholding to divide inliers and outliers can be seen as applying a “top-hat” cost function, according to wich under the threshold (the inliers) have zero cost and above the thershold (the outliers) have constant cost. The RANSAC estimates the model parameters by maximing the number of inliers, in contrast with other methods such as the Least Squares Method. In the presence of outliers, it is a very robust method (Figure 4.21). Thus, this algorithm allows to extract shapes by randomly drawing minimal sets from the point data and constructing the corresponding shape primitives. A minimal set is formed by the smallest number of points required to uniquely define a given type of geometric primitive. The resulting candidate shapes are tested against all points in the data to determine how many of the points are well approximated by the primitive (called the score of the shape). After a given number of trials, the shape which approximates with more points is extracted and the algorithm continues using the remaining data [70].

An example of application of this algorithm could be the identification of the planes in a point cloud scene. In the case of fitting planes to point clouds using RANSAC, many sets of three points are selected at random iteratively, and the one with the largest consensus set is accepted. The points in that consensus set are assigned to the first plane in the scene. Then the points that are contained in it can be removed and then look for the next largest plane. It is possible to find all of the planes in a scene applying this iteratively (Figure 4.22).

RANSAC uses the following steps [66]:

1. Randomly select three non-collinear unique points $\{\boldsymbol{d}_i, \boldsymbol{d}_j, \boldsymbol{d}_k\}$ from the entire point cloud $\mathcal{M}$;

2. Compute the model coefficientes from the three points $(a\boldsymbol{x} + b\boldsymbol{y} + c\boldsymbol{z} + d = 0)$;

3. Compute the distances from all $\boldsymbol{d} \in \mathcal{M}$ to the plane model (a, b, c, d);

4. Count the number of points $\boldsymbol{d^*} \in \mathcal{M}$ whose distance ρ to the plane falls between $0 \leq \rho \leq \delta_t$, where δ_t represente a user specified threshold.

Every set of points $\boldsymbol{d^*}$ is stored, and the above steps are repeated for k iterations, where k is estimated using Equation 4.2. After the algorithm is terminated, the set with the largest number of points (inliers) is selected as the support for each best planar model found. From all $\boldsymbol{d^*} \in \mathcal{M}$, each planar model coefficients are estimated in a least-squares formulation, and a bounding 2D polygon can be estimated as the best approximation given the input data of each plane.

4.2.3 Proposed Algorithm

In this chapter, an approach to robustly compute the 3D geometric features of doors is presented. It combines the explained point cloud processing algorithms and the proposed object detection CNN method. Only a small fraction of the points captured by the RGB-D camera are of interest. In order to efficiently process the point cloud this subset of points i.e. the Region of Interest (ROI) is identified. These regions correspond to those points associated to the 3D representation of doors and handles. To adress this problem, the bounding-boxes provided by the previously presented CNN are extremely useful. Taking into account the nature of the RGB-D sensor the point cloud data could be defined as an RGB image that contains depth information, a direct correspondence between the bounding-boxes and a point cloud region can be stablished. To find this relationship, it is key that the cloud is organized i.e. that it resembles an organized image (or matrix) structure, where the data is split into rows and columns indexed according to its spatial disposition, which is the case (Figure 4.23). For the particular case of the proposed CNN, the boxes are bounded by the pixel coordinates of two opposite corners. The problem is then reduced to determining the relation between the pixel reference system and the point cloud matrix indexes, which is:

$$\text{index} = y \cdot width + x$$

where (x, y) are the pixel coordinates and $width$ is the total number of points along the horizontal axis of the image.

Then, each ROI can be defined as follows:

$$\mathcal{P}_{ROI} = \big\{\mathbf{p}_j \in \mathcal{P} \,|\, (j = width \cdot y + x)\big\} \quad [(x_{min} \leq x \leq x_{max}) \wedge (y_{min} \leq y \leq y_{max})]$$

where: j is the point cloud index; $width$ is the number of pixels contained in the horizontal axis of the image; and (x_{min}, y_{min}) and (x_{max}, y_{max}) represents the coordinates of the upper-left corner and the lower-right corner of the bounding box respectively.

Once the indexes are computed, using the Extract Indices Filter explained in the previous section, each ROI can be segmented, i.e., grouping the point cloud into multiple homogeneous regions (Figure 4.24). Second, in order to avoid further calculation errors, the noisy measurements are filtered for each ROI using the Statistical Outlier Removal filter (Figure 4.25). The number of neighbors to analyze for each point is set to 50, and the standard deviation multiplier to 1, i.e., all points who have a distance larger than 1 standard deviation of the mean distance to the query point will be marked as outliers and removed.

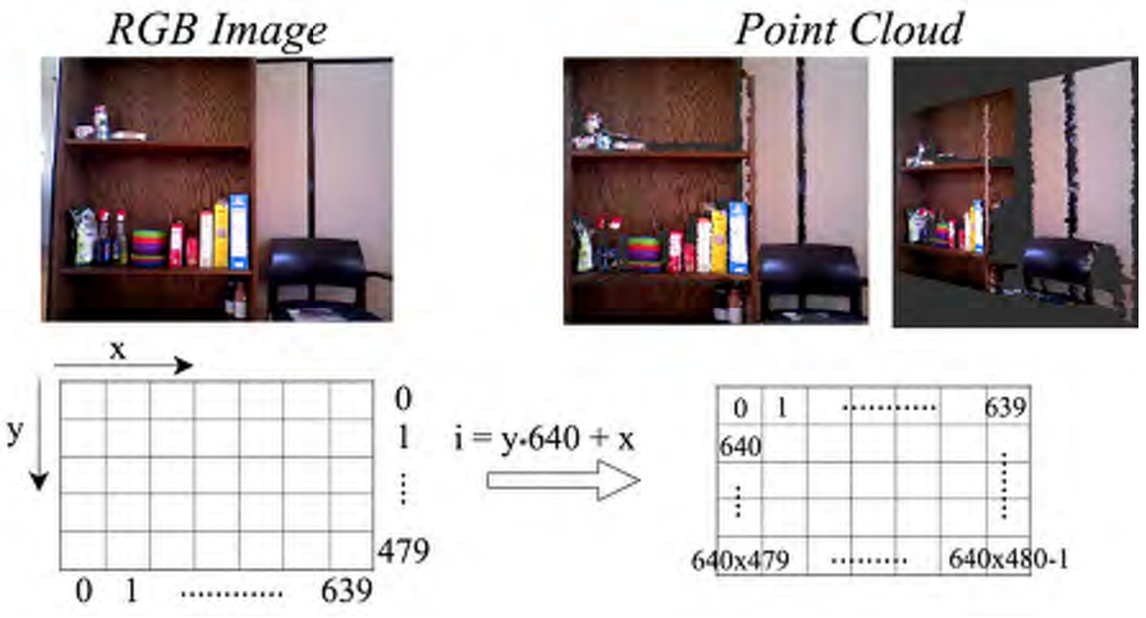


Figure 4.23: From pixel coordinates to point cloud index

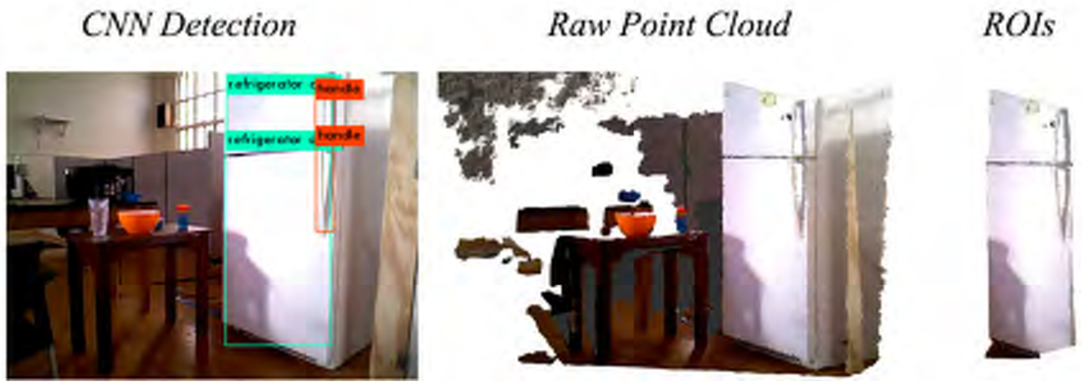


Figure 4.24: ROI extraction

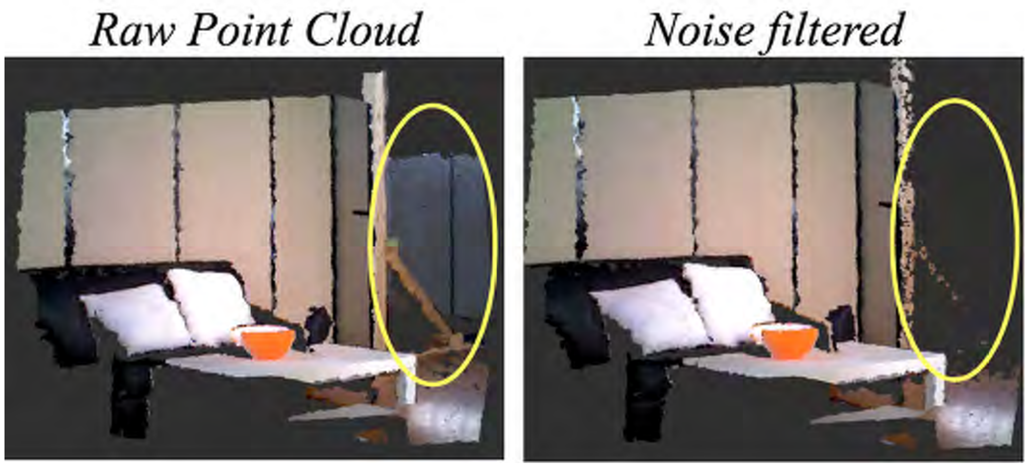


Figure 4.25: Noise Filtering

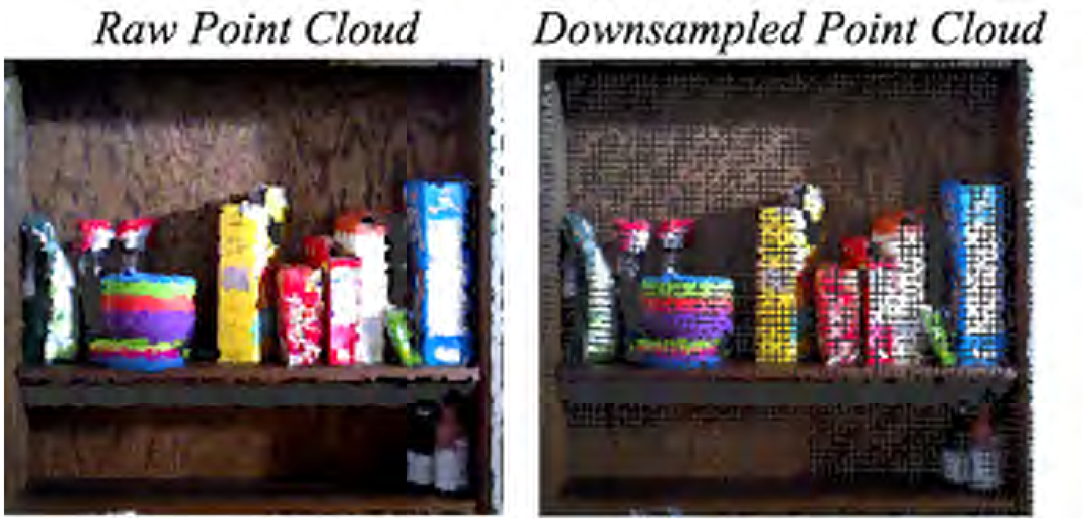


Figure 4.26: Downsampling

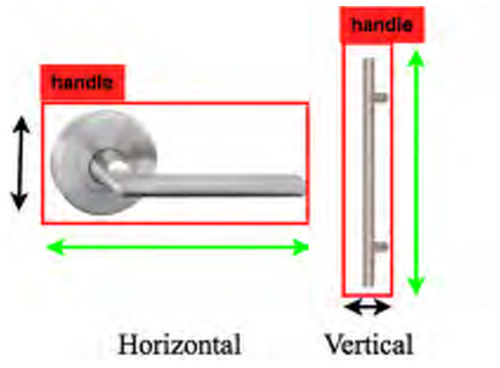


Figure 4.27: Determination of handle orientation

Third, since the objective is to reach the fastest possible processing, the amount of data to be processed should be decreased. However, the geometric features cannot be lost. For this reason, the point cloud is downsampled using the Voxel Grid Filter (Figure 4.26). The size of the voxels used corresponds to a cube of 15mm per side.

At this point the relevant data has been extracted, the measurement errors have been removed and it has been adequately downsampled. Therefore, the geometric features can be computed fast and precisely. To manipulate a door, three of them are key, and for the proposed approach sufficient: the normal direction of the plane defined by the door, the 3D position of the corresponding handle and wether it is orientated vertically or horizontally. For the two first features the RANSAC algorithm can be used since it can work as a classifier to split the data between inliers and outliers and also as an estimator of the parameters of an underlying plane.

- Orientation of the handle: For determining the orientation of the handle, 3D processing techniques could be used. However, if there is no need to compute any precise dimension, only whether it is orientated vertically or horizontally, the dimension of the bounding boxes provided by the CNN are enough. If the width is greater that the height, the handle is in horizontal and vice-versa (Figure 4.27).

- Computation of normal direction to the door plane: when a plane model is fitted using RANSAC, the four components of the plane equation in its Hessian normal form are calculated. Then, it follows: $ax + by + cz = d \;\rightarrow\; \mathbf{n} = (a, b, c)$. Applying the RANSAC algorithm to the ROIs associated to each one of the different classes of detected doors, the normal direction is obtained directly (Figure 4.28).

- Computation of the handle position: The bounding box that contains the handle usually may include some points from the door plane. Also using RANSAC algorithm, these points can be separated, and taking the outliers the resultant cloud only includes those points that correspond to the handle. Assuming the handle position can be represented by the centroid, it is directly calculated from the cloud (Figure 4.29).

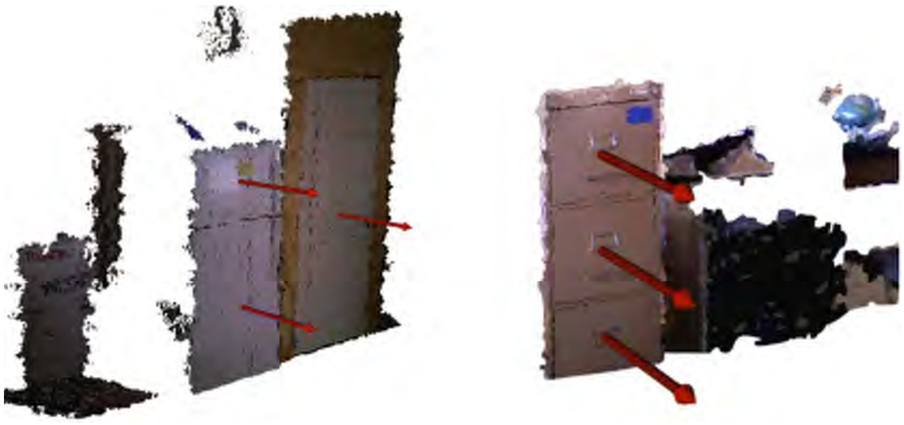


Figure 4.28: Normal vector of different doors

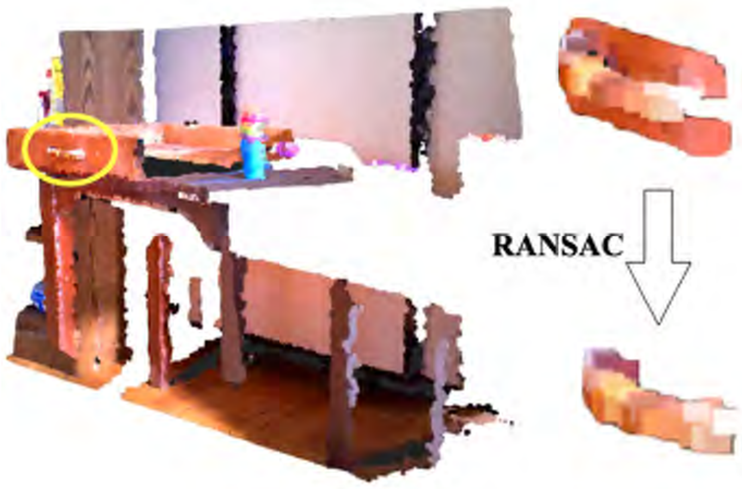


Figure 4.29: Handle point cloud extraction with RANSAC

The proposed algorithm is able to robustly compute the essential 3D geometric features at a speed of 6 fps. This leads to the conclusion that the performance objective of real-time processing has been achieved.

At this point, all the necessary information provided by the perception system for the manipulation of the door has been exploited. However, the robot is still far from being able to reach the handle and grasp it. The perception has to be translated into robot motion that allows the door operation. Robustly manipulating a door involves different robot control algorithms. A description of these methods and how they can adress the proposed problem, will be presented in the next chapter.

5 Manipulation

Moravec's paradox is the discovery by artificial intelligence and robotics researchers that, contrary to traditional assumptions, high-level reasoning requires very little computation, but low-level sensorimotor skills require enormous computational resources. This concept can be easily illustrated with a simple chess game. Chess was one of the great challenge problems of AI. AI researchers eventually developed world-champion-level chess players, except that the computers still need human beings to do the actual moving of the chess pieces. Yes, robots can move chess pieces, but not nearly as well as humans. Championship chess, attained by only the most gifted minds, is actually easier than moving the pieces, which almost every one can easily do [47]. People should not be surprised by the difficulty of manipulation. The creation of autonomous robotic manipulation is surely one of the most challenging engineering problems. It encompasses many difficult problems, involving perception, the robot mechanisms, planning and uncertainty.

Commercially available robotic toys and vacuum cleaners inhabit our living spaces, and robotic vehicles have raced across the desert. These successes appear to foreshadow an explosion of robotic applications in the people daily lives, but without advances in robot manipulation, many promising robotic applications will not be possible. Whether in a domestic setting or at the workplace, the robot needs to physically alter the world through contact. Research on manipulation in human environments may someday lead to robots that work alongside us, extending the time an elderly person can live at home, providing physical assistance to a worker on an assembly line, or helping with household chores [33] (Figure 5.1).

Figure 5.1: Robotic manipulation in domestic environments

To adress the door manipulation problem with robotics systems, different approaches have been proposed [22, 31]. However, no matter the approach or the type of door, it is clear that its operation should involve the following robot manipulation tasks:

1. Grasp the handle.
2. Unlatch the handle.
3. Open the door.

The proposed approach is based on combining motion planning algorithms with prediction and learning capabilities during the execution of the task. The latter will be explained in the following chapter. In this chapter, control algorithms to perform these tasks, as well as how to overcome the manipulation challenges that they represent, will be presented. In section 5.1, how the robot is able to grasp the handle, by combining inverse kynematics and visual perception information, will be explained. In section 5.2, key concepts such as compliance and impedance control as well as a versatile approach to unlatch different types of handles will be presented. And finally, in section 5.3, how the Task Space Region motion planning algorithm can be formulated to solve the highly constrained motion of opening a door will be seen.

5.1 Grasping the Handle

A grasp is commonly defined as a set of contacts on the surface of the object, whose purpose is to constrain the potential movements of the object in the event of external disturbances [43]. The interaction with the enviroment is done by what is known as the robot end-effector, which is usually a device at the end of an arm designed for this purpose. In a wider sense, an end effector can be seen as the part of a robot that interacts with the work environment. In the particular case of the HSR, the end-effector consists of a gripper (Figure 5.2). It is designed to guarantee a stable grasping between a gripper and the object to be grasped, in the project context, the door handle. In order to perform the grasping of the handles and the manipulation of the doors in the unstructured environments of the real world, a robot must be able to perform grasps for the almost unlimited number of different handles it might encounter. Before explaining how these grasps can be achieved, it should be presented how is the end-effector state defined [49].

5.1.1 Pose

A rigid object has six degrees of freedom in a three dimensional space, and its knowledge (i.e. its pose) is required in many robotic applications. Three degrees describe its position and the remaining three describe its orientation. These dimensions behave quite differently. If the value of one of the position dimensions is increased the object will move continuously in a straight line, but if the value of one of the orientation dimensions is increased the object will rotate in some way and soon get back to its original orientation. These two groups of three degrees of freedom must be treated differently [16].

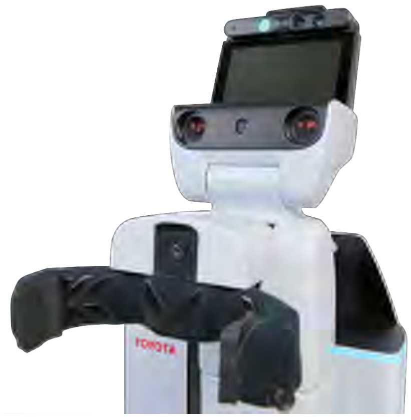


Figure 5.2: HSR end-effector

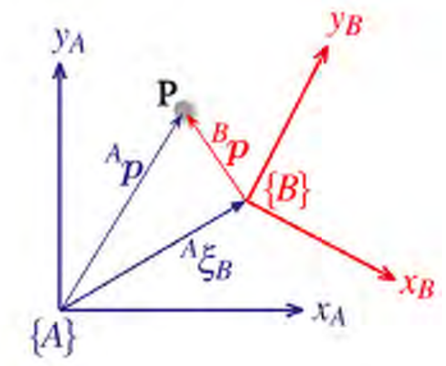


Figure 5.3: The point **P** can be described by coordinate vectors relative to either frame $\{A\}$ or $\{B\}$. The pose of $\{B\}$ relative to $\{A\}$ is ${}^A\xi_B$ [16]

A pose can be seen as a cartesian reference frame, thus, it can be defined in terms of other reference system. The pose of the coordinate frame is denoted by the symbol ξ. Figure 5.3 shows two frames $\{A\}$ and $\{B\}$, and the relative pose ${}^A\xi_B$ which describes $\{B\}$ with respect to $\{A\}$. The leading superscript denotes the reference coordinate frame and the subscript denotes the frame being described. We could also think about ${}^A\xi_B$ as describing some motion, i.e. picking up $\{A\}$ and applying a displacement and a rotation so that it is transformed to $\{B\}$. The point **P** in Figure 5.3 can be described with respect to either coordinate frame by the vectors ${}^A\boldsymbol{p}$ or ${}^B\boldsymbol{p}$ respectively. Formally they are related by ${}^A\boldsymbol{p} =^A \xi_B * {}^B\boldsymbol{p}$, where the right-hand side represents the motion from $\{A\}$ to $\{B\}$ and then to **P**. The operator $*$ transforms the vector, resulting in a new vector that describes the same point but with respect to a different coordinate frame.

There are mainly two different spaces used in kinematics modelling of manipulators called cartesian space and quaternion space. The first one is used to describe translations while the latter is used to describe rotations.

The quaternions are powerful and computational straight-forward and they are widely used in robotics, computer vision and aerospace navigation systems. The quaternion is an extension of the complex number (a hypercomplex number) and it is written as a scalar plus a vector:

$$\boldsymbol{q} = s + \boldsymbol{v} = s + v_1 i + v_2 j + v_3 k$$

where $s \in \mathbb{R}$, $\boldsymbol{v} \in \mathbb{R}^3$ and the orthogonal complex numbers i, j and k are defined as:

$$i^2 = j^2 = k^2 = ijk = -1$$

and it is denoted as:

$$\boldsymbol{q} = s < v_1,\, v_2,\, v_3 >$$

To represent rotations, we are used unit-quaternions denoted by $\mathring{\boldsymbol{q}}$. These are quaternions of magnitude one, that is, those for which $\parallel \boldsymbol{q} \parallel = s + v_1^2 + v_2^2 + v_3^2 = 1$. They can be considered as a rotation of θ about the unit vector $\hat{\boldsymbol{v}}$ and they are related to the quaternion components by:

$$\mathring{\boldsymbol{q}} = \cos\frac{\theta}{2} < \hat{\boldsymbol{v}} \sin\frac{\theta}{2} >$$

Homogeneous transformations based on 4×4 real matrices have often been used within the robotics community. Essentially, all homogeneous transforms that will be used in this work will structured according to the following template [32]:

$$\boldsymbol{T}_A^B = \left[\begin{array}{c|c} Rotation\, Matrix & Posicion\, Vector \\ \boldsymbol{R}_A^B & \boldsymbol{P}_A^B \\ ----------- & ----------- \\ Perspective & Scale \\ \boldsymbol{0}^T & \boldsymbol{1} \end{array}\right]$$

where $\boldsymbol{R}_A^B$ is the relative orientation and $\boldsymbol{P}_A^B$ is the relative position of frame $\{A\}$ relative to frame $\{B\}$. The scale factor will almost always be 1, and the perspective part will be zeros except when modelling cameras.

Under these conditions, it is easy to show, by multiplying the inverse by the original matrix, that the inverse is:

$$\boldsymbol{T}^{-1} = \left[\begin{array}{c|c} \boldsymbol{R}^T & -\boldsymbol{R}^T \cdot \boldsymbol{P} \\ ----------- & ----------- \\ \boldsymbol{0}^T & \boldsymbol{1} \end{array}\right]$$

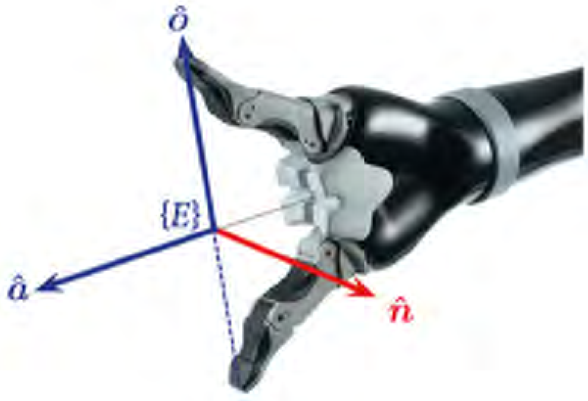


Figure 5.4: Robot end-effector coordinate system defines the pose in terms of an *approach* vector $\hat{\boldsymbol{a}}$ and an *orientation* vector $\hat{\boldsymbol{o}}$, from which the vector $\hat{\boldsymbol{n}}$ can be computed. $\hat{\boldsymbol{n}}$, $\hat{\boldsymbol{o}}$ and $\hat{\boldsymbol{a}}$ vectors correspond to the x, y and z axes respectively of the end-effector coordinate frame [16]

This is very useful to transform a matrix that converts coordinates in one direction (from $\{A\}$ to $\{B\}$) to one that converts coordinates in the opposite direction (from $\{B\}$ to $\{A\}$). The homogeneous transform T_A^B that moves frame $\{A\}$ into coincidence with frame $\{B\}$ (operator) also converts the coordinates (transform) of points in the opposite direction (from frame $\{B\}$ to frame $\{A\}$): ${}^A\boldsymbol{p} = T_A^B \cdot {}^B\mathbf{p}$. Moving a point "forward" in a coordinate system is completely equivalent to moving the coordinate system "backward". This result is usually referred to as operator/transform duality.

For arm-type robots, it is useful to consider a coordinate frame E attached to the end-effector as shown in Figure 5.4. By convention, the axis of the tool is associated with the $z-$axis and is called the approach vector and denoted $\mathbf{a} = (a_x,\, a_y,\, a_z)$. However specifying the direction of the $z-$axis is insufficient to describe the coordinate frame, and the direction of the x- and $y-$axes needs also to be specified. An orthogonal vector to $\mathbf{a}$, that provides orientation, is called the orientation vector, $\mathbf{o} = (o_x,\, o_y,\, o_z)$. These two unit vectors are sufficient to completely define the rotation matrix:

$$R = \begin{pmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{pmatrix}$$

since the remaining first column, the normal vector $\mathbf{n}$, can be computed as: $\mathbf{n} = \mathbf{o} \times \mathbf{a}$.

Any two nonparallel vectors are sufficient to define a coordinate frame. Even if the two vectors $\mathbf{a}$ and $\mathbf{o}$ are not orthogonal they still define a plane and the computed $\mathbf{n}$ is normal to that plane. In this case we need to compute a new value for $\mathbf{o}' = \mathbf{a} \times \mathbf{n}$, which lies in the plane but is orthogonal to each of $\mathbf{a}$ and $\mathbf{n}$.

Defining the transform of a virtual reference frame attached to the end-effector, its state is completely determined. Thus, in order to compute the end-effector grasping pose it is neccessary a translation, an approach vector and an orientation vector.

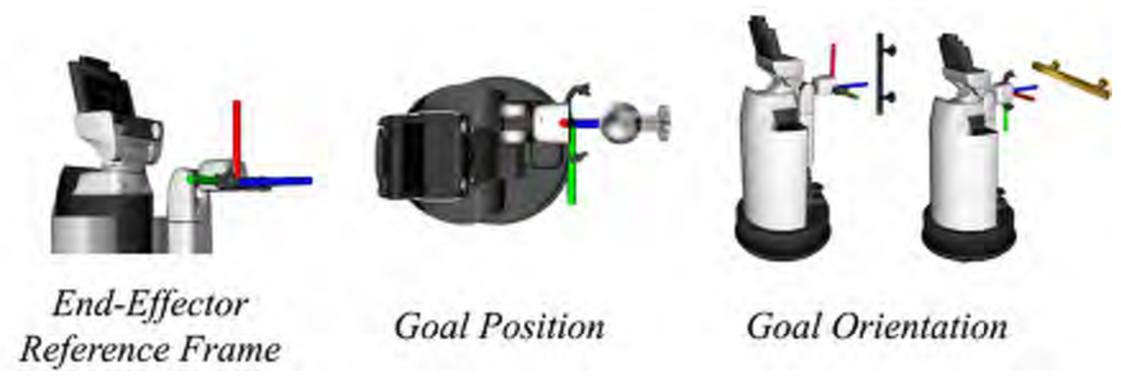


Figure 5.5: The red axis corresponds to the $x-$axis, the green axis to the $y-$axis, and the blue axis to the $z-$axis

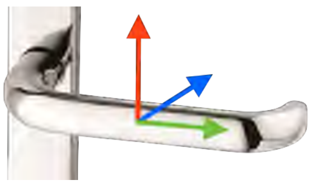


Figure 5.6: Handle reference frame. The color code is the same as in Figure 5.5

5.1.2 Estimating the End-Effector Grasping Pose

In order to grasp the handle, the end-effector goal position can be defined as a few centimeter before reaching the handle and the goal orientation can vary depending on whether the handle is horizontal or vertical. Under these premises, taking into account the definition of the end-effector reference frame for the Toyota HSR, the grasping pose is shown in Figure 5.5. Using the notation of the previous section the $z-$axis coincides with the approaching vector, and the orientation vector can be defined as the $x-$axis.

Determining the end-effector transform could be simplified if it is formulated in terms of a virtual reference frame, that can be defined as the handle reference frame. However, these cartesian reference system cannot be arbitrarily orientated and positioned. A closer look at the goal state in Figure 5.5 shows that the position can be defined as a point in the line determined by the handle centroid and the door normal direction, while the orientation can be defined as a rotation of 0 deg or 90 deg in the $z-$axis direction respect the end-effector reference frame. Therefore, in order to make the goal state transform formulation as easy as possible, the handle reference frame position will be defined as the handle centroid, the $z-$axis as a vector parallel to the door normal pointing inwards its surface and the $x-$axis as a vector perpendicular to the floor plane (it can be either the $x-$ or the $y-$axis, Figure 5.6).

The actual definition of the handle reference frame requires a simplified model of the 3D structure of the door and the handle, and that is where the proposed approach for computing the 3D geometric features, exploiting the robot perception, comes to the scene.

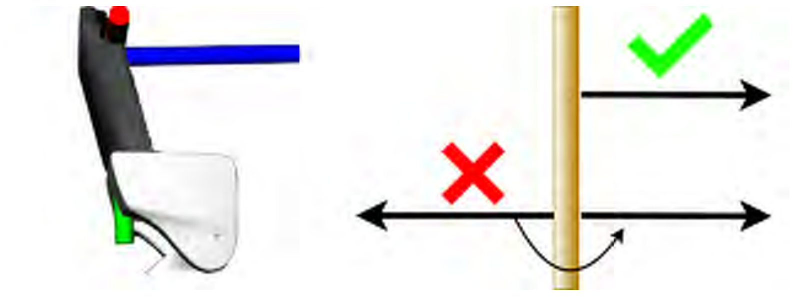


Figure 5.7: The normal vector is forced to point outwards the door from the robot perspective

The information obtained is sufficient to adress this problem. The handle centroid, the door normal direction and the handle orientation are determined. The position of the cartesian reference system can be defined as the centroid. However, for determining the orientation a few more calculations are needed. First, the $z-$ direction can be directly related with the normal vector computed using the RANSAC algorithm. However, it should be remarked that the definition of the normal to a plane does not define the direction where it is pointing univocally. Taking into account that the point cloud is defined in the robot RGB-D reference frame, if it points inwards the door it is sufficient to force the $z-$component to be possitive. If it is not, it would mean that the normal vector is pointing in the wrong direction and therefore, it must be flipped, i.e. multiplied by -1 (Figure 5.7). The other two vectors that define the base must be contained in the door plane. Having a look at the end-effector reference frame goal orientation and assumming that all doors are vertical respect to the plane where the base moves, it is also convenient that one of the vectors of the base is parallel to this direction. It can be either the x- or the y- axis, in the proposed approach it is decided to be the x- axis. In this way, there would not be neccessary to add any rotation to the transformation in case the handle orientation is vertical too. In order to define the vertical direction, the transformation should be defined in terms of the odometry reference frame, that in the case of the HSR, it is always defined with the z- axis perpendicular to the floor plane. Therefore, transforming the centroid and the normal to the odometry reference system, the handle reference frame can be finally defined. Once it is transformed, the $z-$axis is projected into the odometry $x-y$ plane to overcome errors in the normal estimation (Figure 5.8).

The rotation is defined as follows:

$$\begin{pmatrix} 0 & a_y & a_x \\ 0 & -a_x & a_y \\ 1 & 0 & a_z \end{pmatrix}_{odometry}$$

The transform that defines the end-effector goal pose relative to the handle reference frame is simply composed as a translation of $z = -7cm$ and a roll of 90 deg if the handle is horizontal.

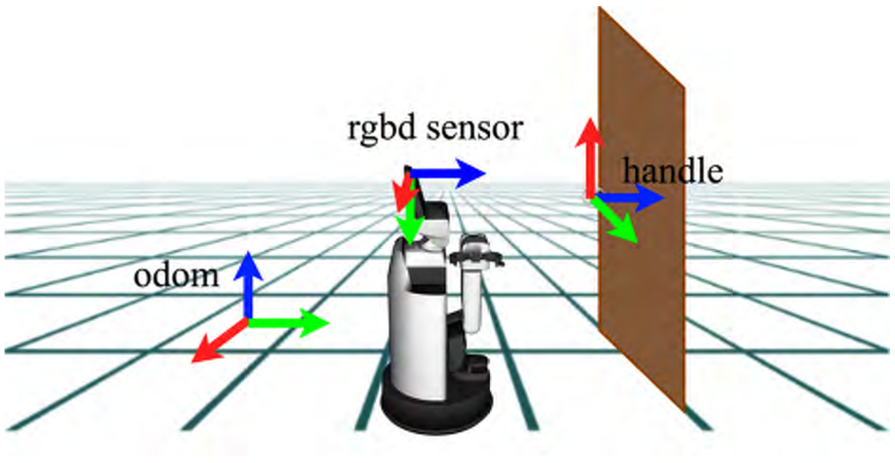


Figure 5.8: Reference frames used to estimate the grasping pose

The overall proposed algorithm for estimating the grasping pose is summarized in Algorithm 5.2:

Algorithm 5.2 End-Effector Grasping Pose Estimation

```
Input: RGB image $\mathcal{I}$ and point cloud $\mathcal{P} = \{\mathbf{p}_j\}_0^{N_{points}}$
Output: Grasping poses $\mathcal{G} = \{\mathbf{g}_k\}_1^{N_{handles}}$ with $\mathbf{g}_k \in SE(3)$
  Bounding boxes $\mathcal{B} = \{b_l\}_1^{N_{objects}} \leftarrow Detect\_Objects(\mathcal{I})$
  for all $b_l \in \mathcal{B}$ do
      $\mathcal{P}_l^{ROI} \leftarrow ROI\_Segmentation(\mathcal{P})$
      $\mathcal{P}_l^{denoised} \leftarrow Remove\_Statistical\_Outliers(\mathcal{P}_l^{ROI})$
      $\mathcal{P}_l^{filtered} \leftarrow Downsample(\mathcal{P}_l^{denoised})$
      if $b_l(class) =$ "$handle$" then
          $orientation_l \leftarrow Bounding\_Box\_Dimensions(b_l)$
          $\mathcal{P}_l^{handle} \leftarrow RANSAC\_Plane\_Outliers(\mathcal{P}_l^{ROI})$
          $\mathbf{O}_l \leftarrow Centroid(\mathcal{P}_l^{handle})$
      else
          Normal $\mathbf{a}_l$; $\mathcal{P}_l^{door} \leftarrow RANSAC\_Plane(\mathcal{P}_l^{filtered})$
          $\mathbf{O}_l \leftarrow Centroid(\mathcal{P}_l^{door})$
      end if
  end for
  $k = 1$
  for all $b_l \in \mathcal{B}$ that $b_l(class) =$ "$handle$" do
      $\mathbf{a}_l \leftarrow Assign\_Door(\mathbf{O}_l)$
      $\mathbf{h}_k \in SE(3) \leftarrow Handle\_Transform(\mathbf{a}_l\,;\,\mathbf{O}_l)$
      $\mathbf{g}_k \leftarrow Goal\_Pose(\mathbf{h}_k\,;\,orientation_l)$
      $k \leftarrow k + 1$
  end forreturn $\mathcal{G}$
```

The approach was tested with the Toyota HSR platform at Anna Hiss Gymnasium in UT at Austin, where a prototype of a home-like arena has been installed to test the performance of service robots in domestic environments. There, a variety of different doors such as drawers, refrigerator, cabinet doors, and room doors are available. There, the robot was able to compute the end-effector grasping pose accurately for multiple handles simultaneously (Figure 5.9), at a processing speed of 6 fps. Thus, it can be concluded that the robot is able to compute the end-effector pose can be estimated in real-time with the proposed approach.

Now, the end-effector goal state is determined, but how can the robot motion be controlled in order to achieve this state? The answer is inverse kynematics.

5.1.3 Inverse Kynematics (IK)

Robot kinematics refers to the analytical study of the motion of a robot manipulator analysing the relationship between the dimensions and connectivity of kinematic chains and the pose (position and orientation) for each link of the manipulator. A mobile manipulator can be schematically represented from a mechanical viewpoint as a kinematic chain of rigid bodies (links) connected by means of joints. One end of the chain is constrained to a base, while an end-effector is mounted to the other end. The resulting motion of the structure is obtained by composition of the elementary motions of each link with respect to the previous one [16]. One systematic way of describing the geometry of a serial chain of links and joints is Denavit-Hartenberg notation. In Denavit-Hartenberg, notation a link defines the spatial relationship between two neighboring joint axes as shown in Figure 5.10. A link is specified by four parameters. The relationship between two link coordinate frames would ordinarily entail six parameters, three each for translation and rotation. For Denavit-Hartenberg notation there are only four parameters but there are also two constraints: axis x_j intersects z_{j-1}, and axis x_j is perpendicular to z_{j-1}. One consequence of these constraints is that sometimes the link coordinate frames are not actually located on the physical links of the robot. Another consequence is that the robot must be placed into a particular configuration (the zero-angle configuration).

An emerging alternative to Denavit-Hartenberg notation is Unified Robot Description Format (URDF), currently the standard ROS XML representation of the robot model. The URDF is an XML specification to describe a robot. The robot model covers the kinematic and dynamic description of the robot, its visual representation and its collision model. Robots in URDF are described by using only two different types of elements: links and joints. A kinematic model is built with a hierarchic structure with a parent-child relationship. This means that, if a joint is rotated around an arbitrary axis, all its children will also rotate around the same axis because they derive all of its parent transformations (Figure 5.11).

In robotics, inverse kinematics (IK) and forward kinematics are two sides of the same coin. On one hand, forward kinematics refers to the use of the kinematic equations of a robot to compute the position of the end-effector from specified values for the joint parameters. On the other hand, IK makes use of the kinematic equations to determine the joint parameters that provide a desired position of the robot end-effector [58].

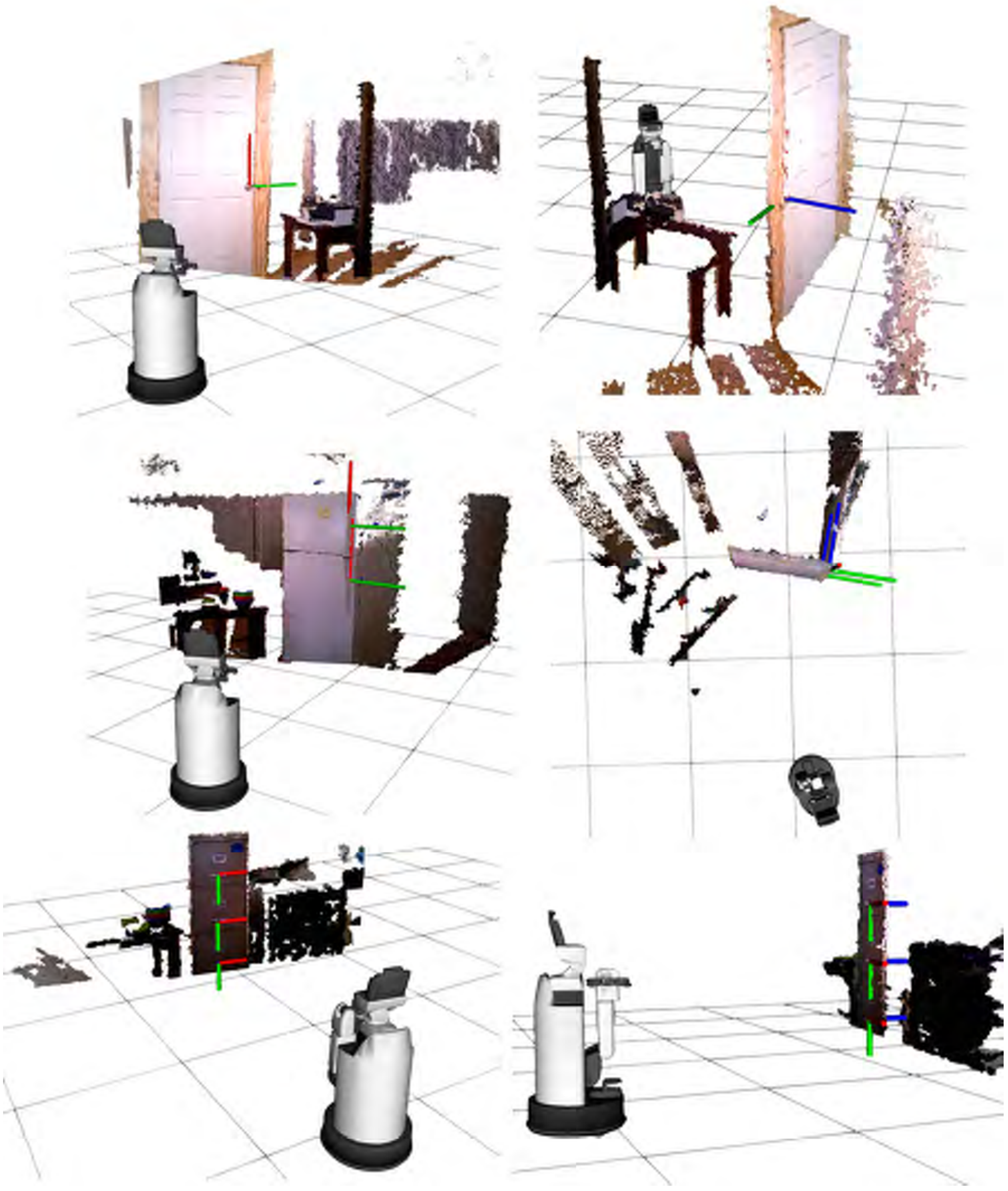
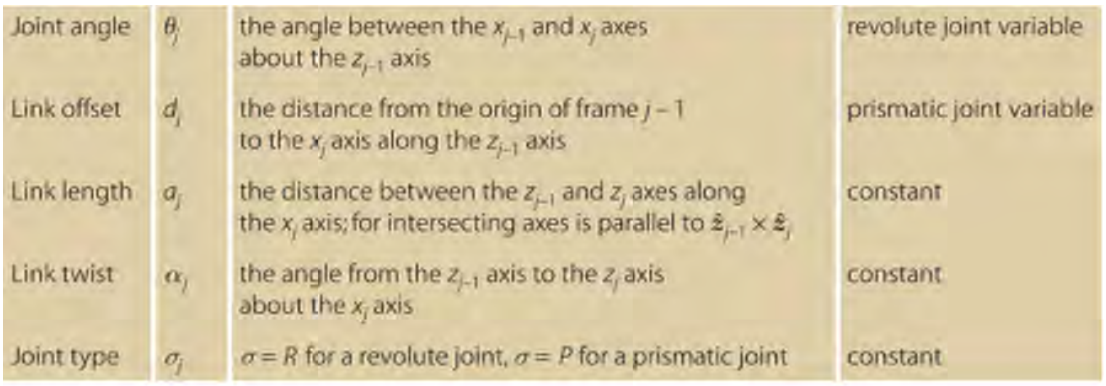


Figure 5.9: Real-time end-effector grasping pose estimation of multiple handles simultaneously, for differente doors



Joint angle	θ_j	the angle between the x_{j-1} and x_j axes about the z_{j-1} axis	revolute joint variable
Link offset	d_j	the distance from the origin of frame $j-1$ to the x_j axis along the z_{j-1} axis	prismatic joint variable
Link length	a_j	the distance between the z_{j-1} and z_j axes along the x_j axis; for intersecting axes is parallel to $\hat{z}_{j-1} \times \hat{z}_j$	constant
Link twist	α_j	the angle from the z_{j-1} axis to the z_j axis about the x_j axis	constant
Joint type	σ_j	$\sigma = R$ for a revolute joint, $\sigma = P$ for a prismatic joint	constant

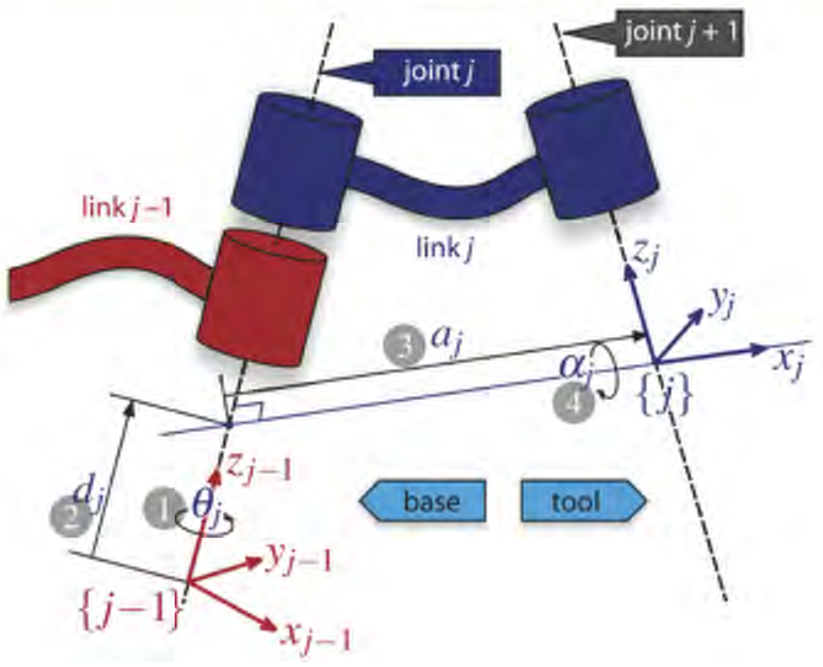


Figure 5.10: Definition of standard Denavit and Hartenberg link parameters. The colors red and blue denote all things associated with links $j-1$ and j respectively. The numbers in circles represent the order in which the elementary transforms are applied. x_j is parallel to $(z_{j-1} \times z_j)$ and, if those two axes are parallel, then the parameter d_j can be arbitrarily chosen. The table at the top summarizes the Denavit-Hartenberg parameters, their physical meaning, and their formal definition [16]

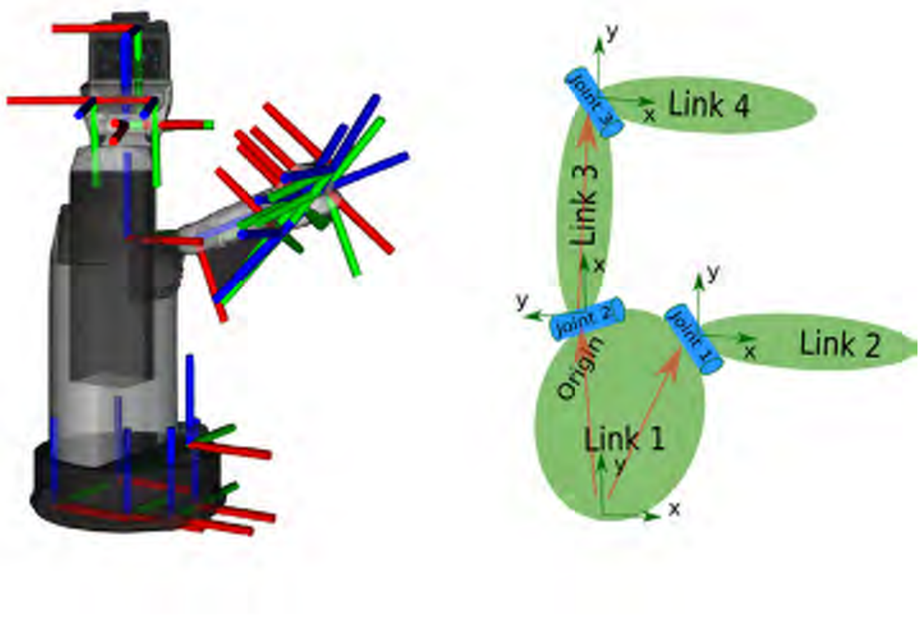


Figure 5.11: URDF hierarchic structure of a set of link elements

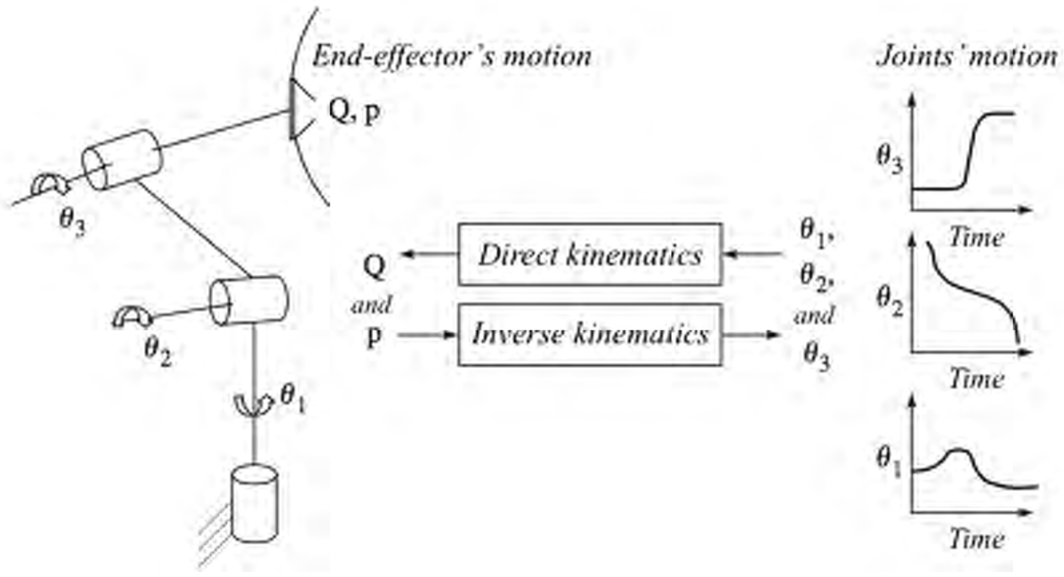


Figure 5.12: Forward and inverse kinematics [69]

Specification of the movement of a robot so that its end-effectors achieve the desired tasks is known as motion planning. Inverse kinematics transforms the motion plan into joint actuator trajectories for the robot (Figure 5.12). Given the equation $\boldsymbol{P} = f(\boldsymbol{\theta})$ for forward kinematics, where $\boldsymbol{P}$ is the current end-effector pose, and $\boldsymbol{\theta}$ is the column vector which representes the joint angles, the inverse kinematics formulation can be derived as the following, where f is a highly non-linear operator which it is difficult to invert:

$$\boldsymbol{\theta} = f^{-1}(\boldsymbol{P})$$

There can be multiple solutions for the IK problem, and it causes that the system has to be able to choose one. The criteria vary but a reasonable choice would be the closest solution from the initial starting point minimizing the amount of energy to move each joint. However, the presence of an obstacle would determine another admissible solution which could not be the closest one. Furthermore, the existence of mechanical joint limits may eventually reduce the number of admissible multiple solutions for the real structure. When there is no analytical solution or it is difficult to find, it might be appropriate to use numerical solutions based on iterative techniques. The iterative methods use the Jacobian matrix which is a linear approximation of kinematics. The Jacobian constitutes one of the most important tools for manipulator characterization. It is useful for finding singularities, analyzing redundancy, solving the inverse kinematic problem, etc. The analytical Jacobian matrix $\boldsymbol{J}$ is a function of the values $\boldsymbol{\theta}$ and is defined as:

$$\boldsymbol{J}(\boldsymbol{\theta}) = \frac{\partial f}{\partial \boldsymbol{\theta}}$$

Each column of $\boldsymbol{J}$ describes an approximated change of the end-effectors position when changing the corresponding joint positions. Then, the forward kinematics problem is formulated as follows:

$$\boldsymbol{P} = f(\boldsymbol{\theta}) \ \rightarrow \ \triangle\boldsymbol{P} = \boldsymbol{J}(\boldsymbol{\theta})\triangle\boldsymbol{\theta}$$

Now, the forward kinematics equation $\triangle\boldsymbol{P}$ is a linear approximation and easier to resolve. By inverting $\boldsymbol{J}(\boldsymbol{\theta})$, the inverse kinematics equation can be written as:

$$\triangle\boldsymbol{\theta} = \boldsymbol{J}^{-\mathbf{1}}(\boldsymbol{\theta})\triangle\boldsymbol{P}$$

where $\triangle\boldsymbol{P} = \boldsymbol{e} = \boldsymbol{T} - \boldsymbol{P}$; $\boldsymbol{e}$ is the desirable change of the end effector; $\boldsymbol{T}$ is the target pose; and $\boldsymbol{P}$ is the current end-effector pose [11].

Given a manipulator and an end-effector pose $\boldsymbol{P}$, a Jacobian matrix can be created and inverted to solve the inverse kinematic problem. However, the solutions are linear approximations of $\boldsymbol{\theta}$ and the equation needs to be done repeatedly until is sufficiently close to a solution. Then the forward kinematics is computed to obtain the new current pose of the end-effector with the new, and check if $\boldsymbol{e}$ is converging to zero. However, in most cases, the Jacobian matrix $\boldsymbol{J}$ may not be square or invertible, and even if it is invertible, $\boldsymbol{J}$ may work poorly as it may be nearly singular.

Several approaches have been proposed to overcome these problems (try to chose $\triangle\boldsymbol{\theta}$ to converge to a solution) and to solve inverse kinematics problems numerically. These methods include Pseudoinverse, Jacobian transpose, the Levenberg-Marquardt Damped Least Squares, quasi-Newton and Conjugate Gradient, Neural Networks and other Artificial Intelligence methods. These clearly have the advantage of being applicable to any kinematic structure, but in general they do not allow computation of all admissible solutions. In practice, to compute IK standard libraries are used. An Inverse Kinematics library is able to solve many robot topologies from tree-like structures to simple chain manipulators with proper performance. The input for this algorithms is commonly the URDF file with the robot structure description and a vector containing the end-effector goal pose.

For this project the Toyota Research Institute (TRI) software has been used to compute the inverse kynematics for the HSR. Since the end-effector grasping pose has already been calculated, applying directly the IK solver the robot is able to reach it (Figure 5.13). Then, it is grasped by simply closing the gripper.

During the tests, the robot was able to grasp correctly the handle of all the three classes of different doors detected. It showed an accurate and robust behavior succeeding on all the experiments. Furthermore, the objective of real-time computation of the previous stages translated on an almost negligible time to estimate the end-effector grasping pose. Thus, it is concluded that the proposed approach for grasping the handle is reliable and efficient.

At this point the robot has been able to exploit the visual information in order to perform a robust grasping of the door handle. However, to be able to open the door, the problem of unlatching the handle must be adressed first.

5.2 Unlocking the Handle

Opening the door requires an adequate manipulation of the handle. There exists a high variety of mechanisms to open a door. Some of them do not require any specific actuation while others generally require a rotation to be applied. The resolution of the HSR vision system does not allow the perception of detailed features needed to infere the particular kynemathic model of the handle such as the attaching points to the door or the axis of rotation.

Defining a general strategy to open a handle should rely in some sensing capability which allows to differentiate between the different mechanisms. The HSR has a six axis force sensor in the wrist (Figure 5.14) that can be useful for this purpose.

When a robot manipulator makes contact with the environment, control of both force and motion is required. The unification of these two objectives is known as “compliance” or “impedance control”. These concepts will be key for unlocking the door handle but also for the next chapter proposed approach.

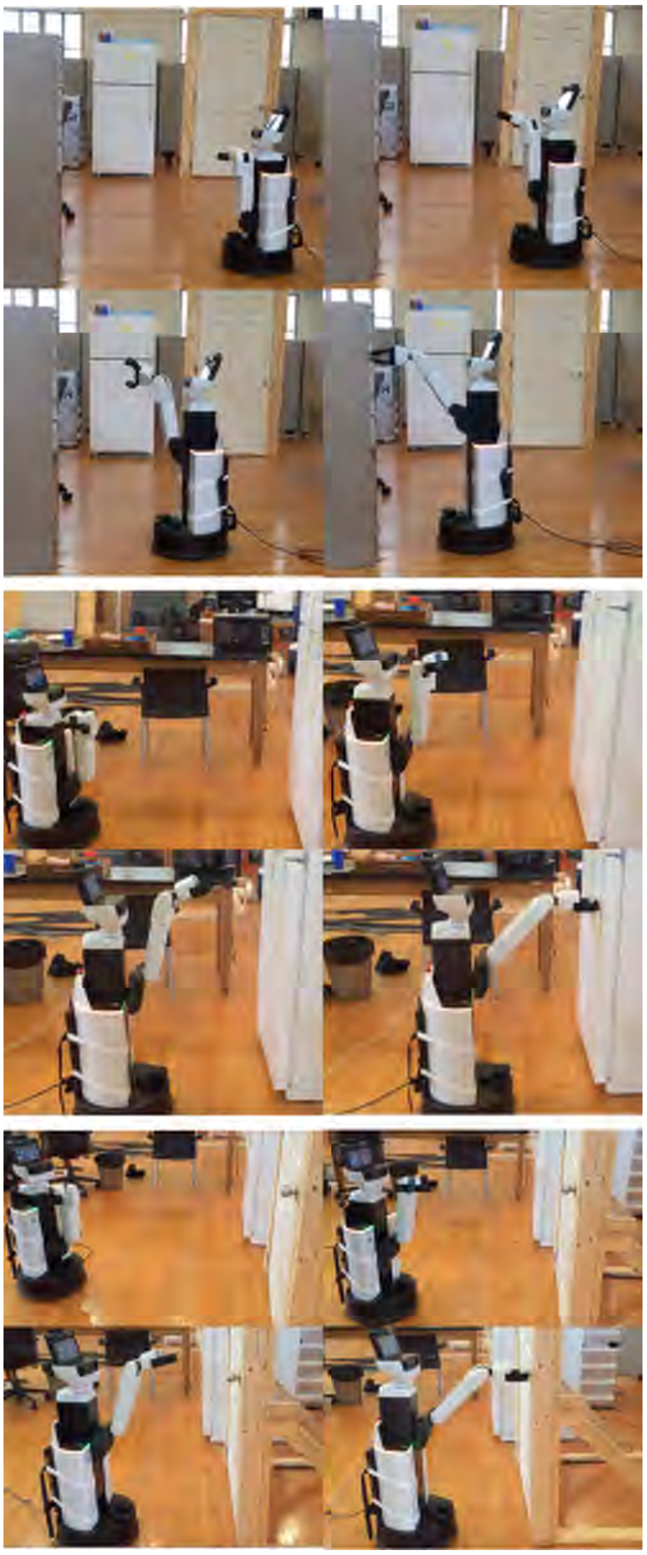


Figure 5.13: HSR grasping the handles of different doors after solving the inverse kinematics for the estimated goal pose

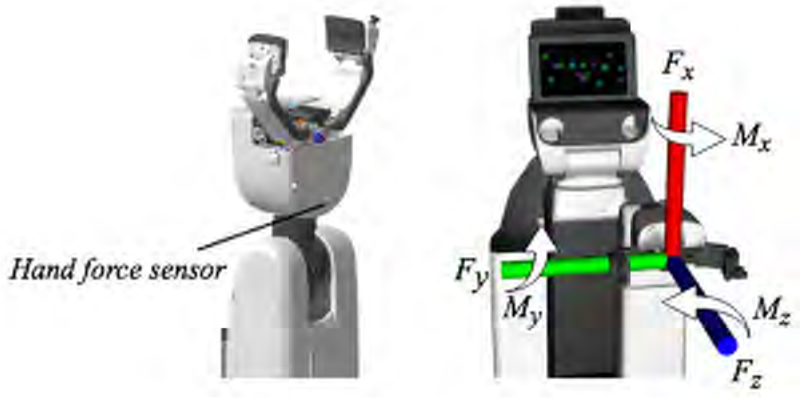


Figure 5.14: HSR six axis hand force sensor

5.2.1 Impedance Control

Simple tasks may need only trajectory control where the robot end-effector is moved merely along a prescribed time trajectory. However, a number of complex tasks, including manipulation of some objects, entail the control of physical interactions and mechanical contacts with the environment. Achieving a task goal often requires the robot to comply with the environment, react to the force acting on the end-effector, or adapt its motion to uncertainties of the environment.

A very common control structure is the nested control loop. The outer loop is responsible for maintaining position and determines the velocity of the joint that will minimize position error. The inner loop is responsible for maintaining the velocity of the joint as demanded by the outer loop. Consider the motor which actuates the j^{th} revolute joint of a serial-link manipulator. From Figure 5.10 we recall that joint j connects link $j-1$ to link j. The motor exerts a torque that causes the outward link, j, to rotationally accelerate but it also exerts a reaction torque on the inward link $j-1$. Gravity acting on the outward links j to N exert a weight force, and rotating links also exert gyroscopic forces on each other. The inertia that the motor experiences is a function of the configuration of the outward links. The situation at the individual link is quite complex but for the series of links the result can be written concisely as a set of coupled differential equations in matrix form:

$$\boldsymbol{Q} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{F}(\dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{J}(\boldsymbol{q})^T\boldsymbol{W} \tag{5.1}$$

where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ are respectively the vector of generalized joint coordinates, velocities and accelerations, $\boldsymbol{M}$ is the joint-space inertia matrix, $\boldsymbol{C}$ is the Coriolis and centripetal coupling matrix, $\boldsymbol{F}$ is the friction force, $\boldsymbol{G}$ is the gravity loading, and $\boldsymbol{Q}$ is the vector of generalized actuator forces associated with the generalized coordinates $\boldsymbol{q}$. The last term gives the joint forces due to a wrench $\boldsymbol{W}$ applied at the end-effector and $\boldsymbol{J}$ is the manipulator Jacobian.

This equation describes the manipulator rigid-body dynamics and is known as the inverse dynamics: given the pose, velocity and acceleration it computes the required joint forces or torques [16].

Force and compliance controls are fundamental strategies for performing tasks entailing the accommodation of mechanical interactions in the face of environmental uncertainties. To define what compliance is, the definition of non-compliance is useful. A non-compliant (stiff) robot end effector is a device which is designed to have predetermined positions or trajectories. No matter what kind of external force is exerted the robotic end effector will follow the exact same path each and every time. On the other hand, a compliant end effector can reach several positions and exert different forces on a given object. Compliance aims towards either human safety (passive) or process improvement (active).

Impedance control is a unified control scheme suitable for dealing with the mechanical interaction tasks incorporating contact processes. The impedance refers to the dynamic relationship between the motion variables of manipulator and the contact force. The goal is to control this relationship as desired to fulfill the requirements of a specified manipulation task, such as keeping the position of robot end-effector and the contact force under preset safe ranges simultaneously [72]. The main merit of impedance control is that it provides an effective way to control the motion and contact force at the same time by adjusting a relationship between position and force in the task space. Rather than totally separating the task space into subspaces of either position or force control, compliance control reacts to the endpoint force such that a given functional relationship, typically a linear map, is held between the force and the displacement. Namely, a functional relationship to generate is given by:

$$\triangle \boldsymbol{P} = \boldsymbol{C}\boldsymbol{F}$$

where $\boldsymbol{C}$ is an $m \times m$ compliance matrix, and $\triangle \boldsymbol{P}$ and $\boldsymbol{F}$ are endpoint displacement and force represented in an $m-$dimensional task coordinate system. Note that the inverse to the compliance matrix is a stiffness matrix: $\boldsymbol{K} = \boldsymbol{C}^{-1}$, if the inverse exists [5]. The components of the compliance matrix, or the stiffness matrix, are design parameters to be determined in such a way that they meet task objectives and constraints.

Opening a door, for example, can be performed with the compliance illustrated in Figure 5.15. The trajectory of the door knob is geometrically constrained to the circle of radius R centered at the door hing. The robot hand motion must comply to the constrained doorknob trajectory, although the trajectory is not exactly known. This task requirement can be met by assigining a small stiffness, i.e. a high compliance, to the radial direction perpendicular to the trajectory. As illustrated in Figure 5.15, such a small spring constant generates only a small restoring force in response to the discrepancy between the actual handle trajectory and the reference trajectory of the robot hand. Along the direction tangent to the doorknob trajectory, on the other hand, a large stiffness, or a small compliance, is assigned. This is to force the handle to move along the trajectory despite friction and other resistive forces. The stiffness matrix is therefore given by:

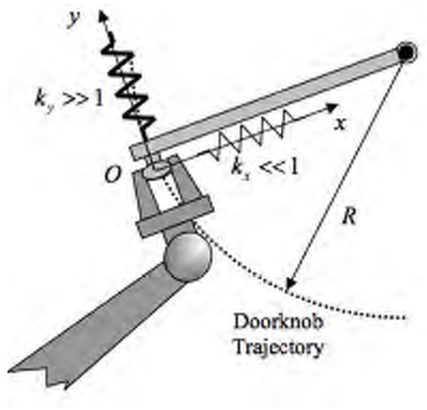


Figure 5.15: Door opening with compliance control [5]

$$\boldsymbol{K} = \begin{pmatrix} k_x & 0 \\ o & k_y \end{pmatrix};\ k_x \ll 1,\ k_y \gg 1$$

with reference to the task coordinate system $O - xy$.

There are multiple ways of synthesizing a compliance control system. Compliance synthesis is trivial for single joint control systems. For general n degree-of- freedom robots, however, multiple feedback loops must be coordinated. We now consider how to generate a desired $m \times m$ compliance or stiffness matrix specified at the endpoint by tuning joint feedback gains. Let $\boldsymbol{J}$ be the Jacobian relating endpoint velocity, $\dot{\boldsymbol{P}} \in \mathbb{R}^{m\times 1}$, to joint velocities, $\dot{\boldsymbol{q}} \in \mathbb{R}^{n\times 1}$, and $\boldsymbol{\tau} \in \mathbb{R}^{n\times 1}$, be joint torques associated with joint coordinates $\boldsymbol{q}$. Let $\boldsymbol{K}_P$ be a desired endpoint stiffness matrix defined as:

$$\boldsymbol{F} = \boldsymbol{K}_P \triangle \boldsymbol{P}\,. \tag{5.2}$$

Using the Jacobian and the duality principle as well as Eq. 5.2: $\boldsymbol{\tau} = \boldsymbol{J}^T\boldsymbol{F} = \boldsymbol{J}^T\boldsymbol{K}_P\triangle\boldsymbol{P} = \boldsymbol{J}^T\boldsymbol{K}_P\boldsymbol{J}\cdot\triangle\boldsymbol{q}$, which can be expressed as follows: $\boldsymbol{K}_q = \boldsymbol{J}^T\boldsymbol{K}_P\boldsymbol{J} \ \Rightarrow \boldsymbol{\tau} = \boldsymbol{K}_q\triangle\boldsymbol{q}$. This implies that the necessary condition for joint feedback gain $\boldsymbol{K}_q$ to generate the endpoint K is given by: $\boldsymbol{K}_q = \boldsymbol{J}^T\boldsymbol{K}_P\boldsymbol{J}$ or, in other words, that $\boldsymbol{K}_q$ is the joint feedback gain matrix [5].

From this developement it can be seen that the complexity of the problem when dealing with n degrees of freedom increases considerably respect to the case of one degree of freedom. For these reason, usually only certain joints incorporate this kind of control. All in all, compliance control can be seen as a movement control that allows a certain error tolerance in the desired trajectory when interaction forces appear. These principles can be applied to adress the problem of unlocking the door handle.

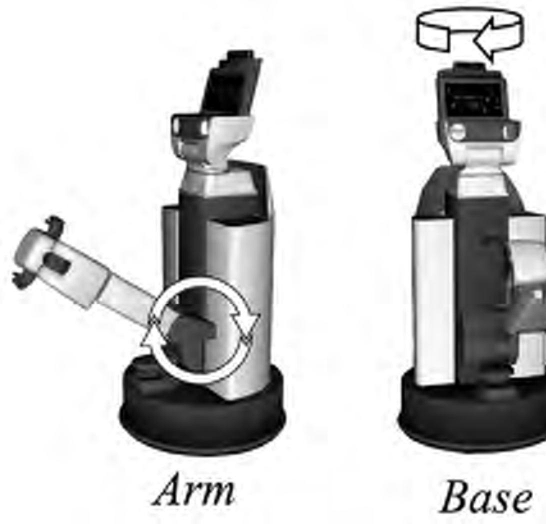


Figure 5.16: HSR joints with impedance control

5.2.2 Versatile Approach to Operate Different Kinds of Handles

The HSR can follow the force applied to the hand by utilizing the 6 axis force sensors mounted on the wrist. The Toyota Research Institute has incorporated a few different modes on impedance control in the degrees of freedom i.e. joints, shown in 5.16. Since there is not prior knowledge about the handle mechanism, the manipulation strategy could be to deduce it using the lectures from the force sensor. It should be noted that these readings correspond to reaction forces, thus, it means that an actuation will be needed in order to infere how the handle is unlocked. Since there is uncertainty in the manipulation, this problem can be adressed implementing a simple impedance control on the wrist.

The HSR motion is controlled with position commands. Therefore, a simple impedance control algorithm could be to move or stop moving when the forces detected by the force-sensor are below or above a certain threshold. In this way, the force applied is controled with position commands. The strategy can be summarized as "if it moves, it works". The proposed approach consists then, in the following procedure: a position command is used to turn the wrist joint anti-clockwise, if the lecture of the torque in the z direction is higher than a certain threshold, the position command is canceled. Then, this is repeated turning the wrist in the other direction. If the wrist can turn the input angle without applying a torque above of the threshold it succeeds. After the last step of after it succeeds, the robot pulls to start the opening process (Figure 5.17). In this project only doors that open by pulling them have been considered. However, with the proposed approach, in the same way that is determined whether the handle mechanism requires a rotation and in that case, in which direction, using the lecture of the force in the $z-$axis it could be determined if the door is opened by pulling or pushing.

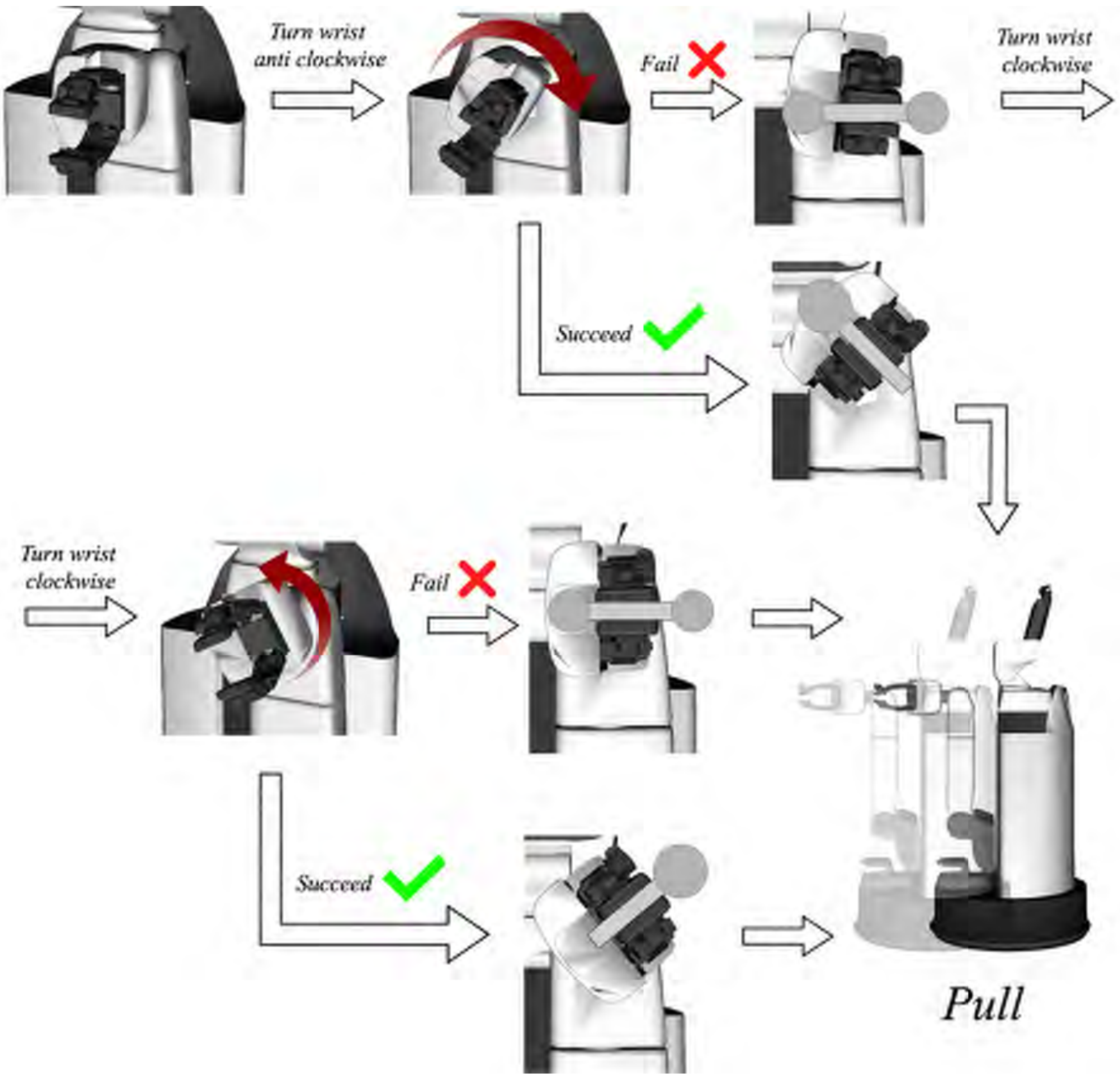


Figure 5.17: Verstatile approach to operate handles

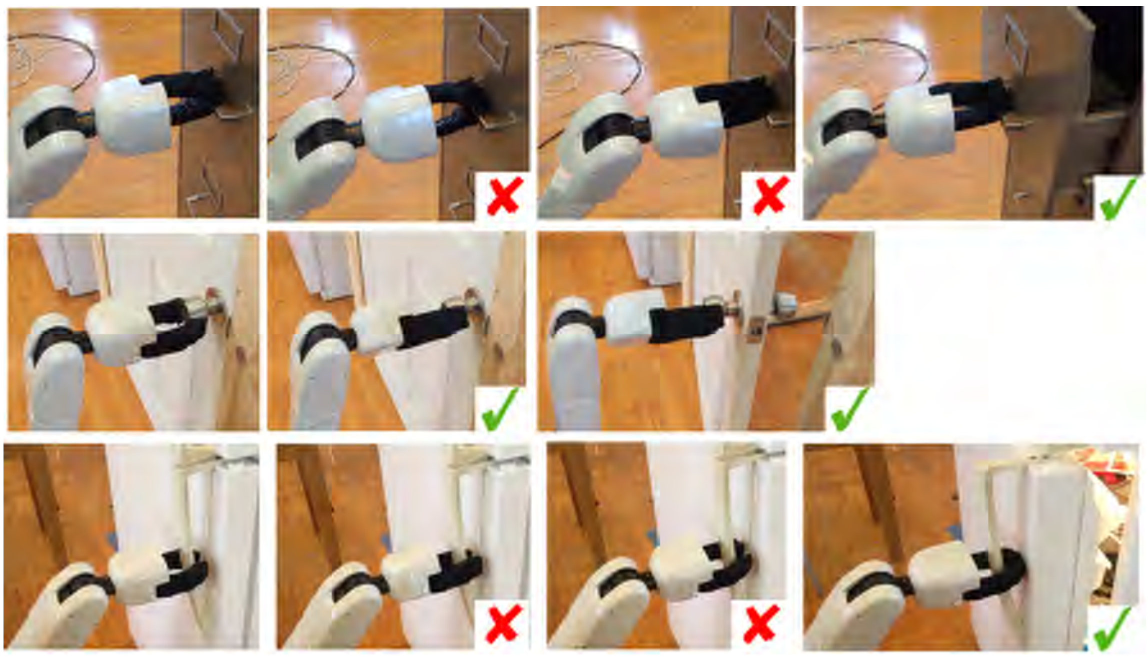


Figure 5.18: HSR unlocking different types of handles

The proposed approach was tested obtaining satisfactory results (Figure 5.18). The robot was able to unlock the different handles available in the laboratory with different mechanisms using the same algorithm.

At this point, the only remaining step is to open the door following the corresponding trajectory. Impedance control for multiple degrees of freedom is very complex and for a robot such as the Toyota HSR Robot almost beyond the scope of this project. Futhermore, it would require a sensing capability that these platform does not have since only a force sensor mounted on the wrist might not provide the neccesary force feedback to achieve a good compliance control. The available compliance can be very useful but not sufficient. For these reasons, additional algorithms should be contemplated for this last step.

5.3 Door Opening Motion

Computing a motion that enables a mobile manipulator to open a door is challenging because it requires tight coordination between the motions of the arm and the base. This makes the problem high-dimensional and thus hard to plan for since it is highly constrained. On the other hand, hard-coding motion plans “a priori” is even harder due to high variability in the conditions under which the doors may need to be opened [13]. To adress this problem, an efficient motion planning under constraints is needed. But before presenting the proposed approach, it is neccesary a glimpse into the field of motion planning in robotics.

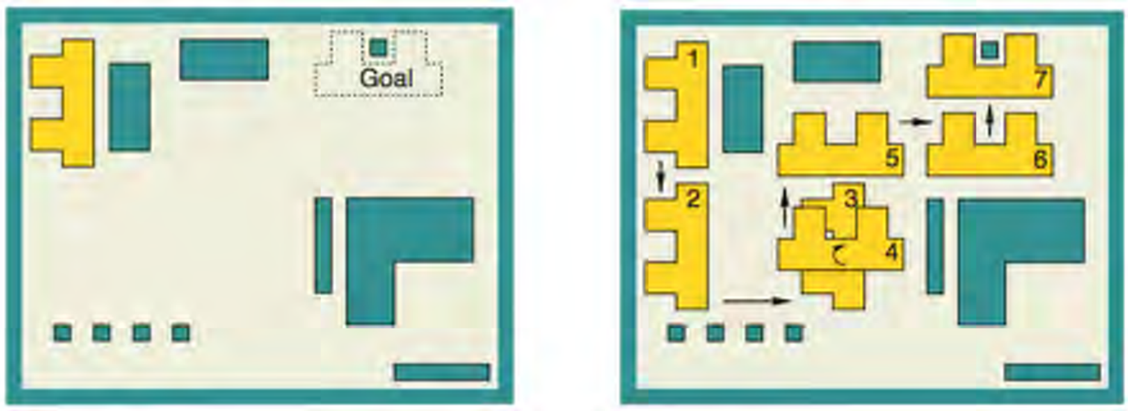


Figure 5.19: Basic motion planning problem [40]

5.3.1 Motion Planning

Motion planning involves such diverse aspects as computing collision-free paths among possibly moving obstacles, planning sliding and pushing motions to achieve precise relations among objects, reasoning about uncertainty to build reliable sensory-based motion strategies, dealing with models of physical properties such as mass, gravity and friction, and planning stable grasps of objects. Therefore, motion planning requires the robot to consider geometrical constraints, as well as physical and temporal constraints. In addition, uncertainty may require that it plans not only motion commands, but also their interaction with sensing [38]. Given the emphasis on autonomous mobile manipulation in the context of this project, the focus will be on motions in service of manipulation, i.e., collision-free motion for end-effector placement. The problem of generating such motion is a specific instance of the motion planning problem.

5.3.1.1 Basic Motion Planning Problem

In the basic motion planning problem definition, the fundamental assumptions are that the robot is the only moving object in the workspace and the dynamic properties of the robot are ignored, thus avoiding temporal issues. The motions are also restricted to non-contact motions, so that the issues related to the mechanical interaction between two physical objects in contact can be ignored. These assumptions essentially transform the "physical" motion planning problem into a purely geometrical path planning problem. Geometrical issues are simplified by assuming that the robot is a single rigid object. Thus, motions of this object are only constrained by the obstacles [38].

Let W denote the world that contains the robot and obstacles (Figure 5.19). For a two-dimensional (2-D) world, $\mathcal{W} = \mathbb{R}^2$ and $\mathcal{O} \subset \mathcal{W}$is the obstacle region, which has a piecewise-linear (polygonal) boundary (the complement $\mathcal{W}/\mathcal{O}$ is assumed to be a bounded open set). The robot is a rigid polygon that can move through the world but must avoid touching the obstacle region. For a three-dimensional (3-D) world, the only differences are that $\mathcal{W} = \mathbb{R}^3$, and $\mathcal{O}$ and the robot are defined with polyhedra instead of polygons.

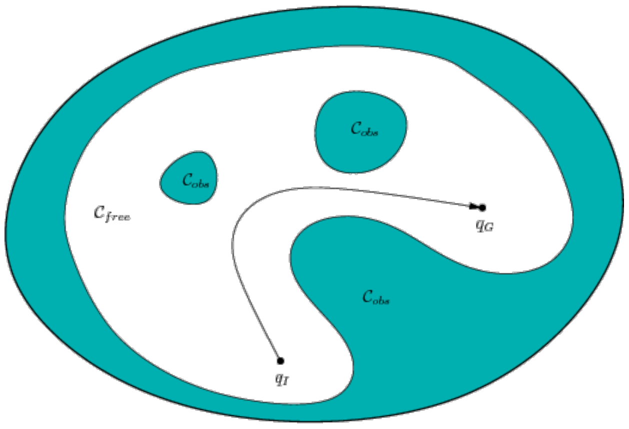


Figure 5.20: Basic motion planning problem in the configuration space [39]

In terms of algorithms inputs and outputs, the input of a motion plannning algorithm would be an initial placement of the robot, a desired goal placement, and a geometric description of the robot and obstacle region. The output would be a precise description of how to move the robot gradually from its initial placement to the goal placement while never touching the obstacle region [40]. The geometric path is normally defined in the operating space (or workspace) of the robot, because the task to be performed, as well as the obstacles to avoid, are described in the operating space more naturally than in the joint space (or configuration space). Planning the trajectory in the operating space is usually done when the motion follows a path with specific geometric characteristics defined in the operating space. However, in most cases the trajectory is planned in the joint space of the robot because, since the control action on the manipulator is made on the joints, planning in the operating space requires a kinematic inversion to transform the end-effector position and orientation values into the joint values.

5.3.1.2 Configuration Space

The configuration space is the set of all possible transformations of the robot. The number of degrees of freedom of a robot system is the dimension of the set of transformations, or the minimum number of parameters needed to specify a configuration. Physical concepts, such as force and friction, can also be represented in this space as additional geometrical constructs. The introduction of the configuration space reduced the search for an obstacle-free path to computing a continuous path for a point from start to goal that avoids the forbidden regions representing the physical static obstacles.

Configuration space is essentially a representational tool for formulating motion planning problems precisely. It is an important abstraction that allows to use the same motion planning algorithm to a problem that differs in geometry and kinematics. Thus, path planning becomes a search on a space of transformations. The basic motion planning can be expressed in the configuration space of the robot in the following way (Figure 5.20) [39]:

- Given robot $\mathcal{A}$ and obstacle $\mathcal{B}$ models, configuration space (C-space, $\mathcal{C}$), and points $q_I, q_G \in \mathcal{C}_{free} \Rightarrow$ Automatically compute a path $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$ so that $\tau(0) = q_I$ and $\tau(1) = q_G$.
- Thas is, a motion planning algorithm must find a path in the C_{free} space (free region) from an initial configuration to a goal configuration.

The work in the configuration space requires preliminary computation of $\mathcal{C}_{obs}$ (obstacle region) and $\mathcal{C}_{free}$, defined in the following terms ($\mathcal{C} = \mathcal{C}_{obs} + \mathcal{C}_{free}$):

$$\begin{cases} \mathcal{C}_{obs} : \text{ closed set} \\ \mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap B \neq \emptyset\} \\ C_{obs} = \text{set of disallowed configurations} \end{cases}$$

$$\begin{cases} \mathcal{C}_{free} : \text{ open set} \\ \mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs} \\ C_{free} = \text{set of allowed (feasible) configurations} \end{cases}$$

The computation of $\mathcal{C}_{obs}$ can be exact (algebraic model needed) or approximate (grid decomposition or sampling scheme).

5.3.1.3 Constraints on Configuration

Constraints involving the pose of a robot end-effector are some of the most common constraints in manipulation planning. Although ubiquitous, these constraints present significant challenges for planning algorithms. Many types of constraints can limit a robot motion. This analysis focuses on scleronomic (time-invariant) holonomic constraints, which are time-invariant constraints evaluated at a given configuration of the robot [7]. Let the configuration space of the robot be $\mathcal{C}$. A path in that space is defined by $\tau : [0, 1 \rightarrow \mathcal{C}]$. Thus, a path is a curve in the configuration space represented either by mathematical expression, or a sequence of points. It is interesting to note, at this point, that a trajectory is nothing more than a path on which a timing law is specified, that is, a path with an allocation of time at each point along the path. Constraints evaluated as a function of a configuration $q \in \mathcal{C}$ in τ are considered. The location of q in τ determines which constraints are active at that configuration. Thus, a constraint is defined as the pair $\{\mathcal{Q}(q), s\}$, where $\mathcal{Q}(q) \in \mathbb{R} \geq 0$ is the constraint-evaluation function. $\mathcal{Q}(q)$ determines whether the constraint is met at that q, and $s \subseteq [0, 1]$ is the domain of this constraint, that is, where the constraint is active in the path τ. Then, to say that a given constraint is satisfied we require that $\mathcal{Q}(q) = 0 \ \forall q \in \tau(s)$. Each constraint defined in this way implicitly defines a manifold in $\mathcal{C}$ where $\tau(s)$ is allowed to exist. Thus, given a constraint, the manifold of configurations that meet this constraint, $\mathcal{M}_{\mathcal{Q}} \subseteq \mathcal{C}$, is defined as:

$$\mathcal{M}_{\mathcal{Q}} = \{q \in \mathcal{C} : \mathcal{Q}(q) = 0\}$$

In order for τ to satisfy a constraint, all the elements of $\tau(s)$ must lie within $\mathcal{M}_\mathcal{Q}$. Conversely, if $\exists\, q \notin \mathcal{M}_\mathcal{Q}$ for $q \in \tau(s)$, then τ is said to violate the constraint. In general, we can define any number of constraints for a given task, each with their own domain. Let a set of n constraint-evaluation functions be $\mathcal{Q}$, and the set of domains corresponding to those functions be $\mathcal{S}$. Then we define the constrained path planning problem as:

$$\text{find } \tau \,:\, q \in \mathcal{M}_{\mathcal{Q}_i} \begin{cases} \forall q \in \tau(S_i) \\ \forall i \in \{1, ..., n\} \end{cases}$$

Note that the domains of two or more constraints may overlap, in which case an element of τ may need to lie within two or more constraint manifolds.

The main issue that makes solving the constrained path planning problem difficult is that constraint manifolds are difficult to represent. There is no known analytical representation for many types of constraint manifolds (including pose constraints) and the high-dimensional C-spaces of most practical robots make representing the manifold through exhaustive sampling prohibitively expensive. To adress the problem proposed in this project, where a highly constrained motion planning is required, the Task Space Region (TSR) framework is adopted.

5.3.2 Task Space Region (TSR): Pose Constrained Manipulation

Some of the most successful methods in control theory for manipulation have come from controllers that seek to minimize a given function in the neighborhood of the robot current configuration through gradient-descent. These controllers can succeed or fail depending on the prioritization of constraints and it is unclear which of the multiple simultaneous constraints should be prioritized ahead of others. Sampling-based planners are mos useful to provide a practical solution to the problem of global planning for manipulation tasks, especially for high-DOF manipulators. Sampling-based manipulation planners are designed to explore the space of solutions efficiently, without the exhaustive computation required for dynamic programming and without being trapped by local minima like gradient-descent controllers. The main idea of sampling-based planning is avoid constructing explicity the $\mathcal{C}_{obs}$ space, through a sampling scheme. With this approach, planning algorithms are independent of the particular geometric models [39].

Task Space Region (TSR) is a constrained manipulation framework, designed to be used with sampling-based planners, with a specific constraint representation that has been developed for planning paths for manipulators with end-effector pose constraints [7]. TSRs describe end-effector constraint sets as subsets of $SE(3)$ (Special Euclidean Group). These subsets are particularly useful for specifying manipulation tasks ranging from reaching to grasp an object and placing it on a surface or in a volume, to manipulating objects with constraints on their pose such as transporting a glass of water. TSRs are not intended to capture every conceivable constraint on pose. Instead they are meant to be simple descriptions of common manipulation tasks that are useful for planning.

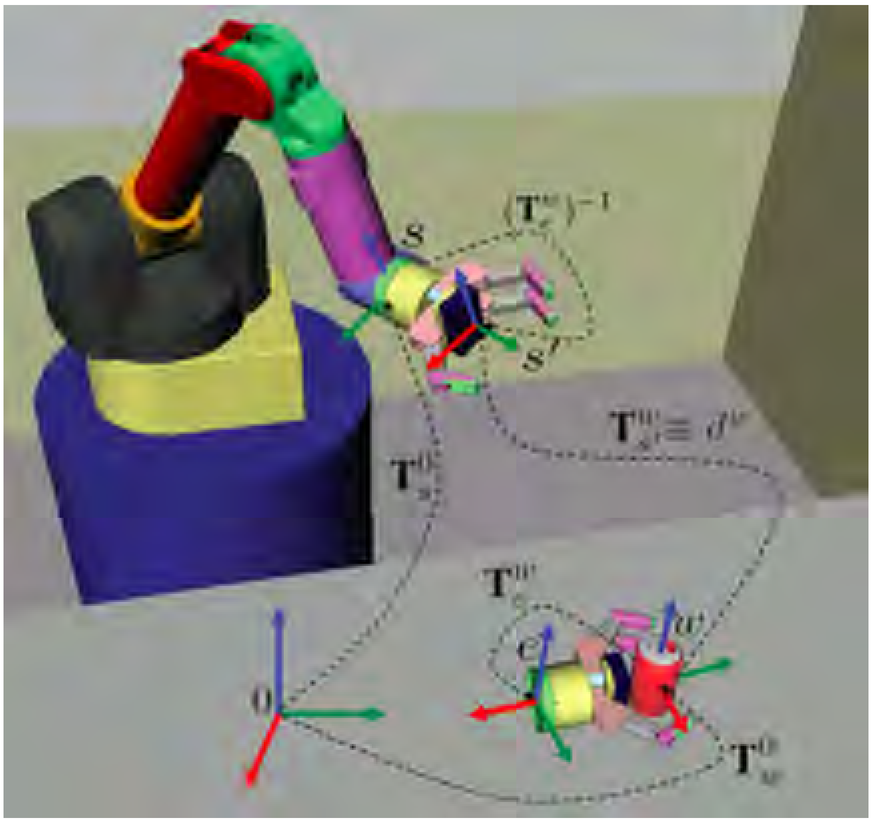


Figure 5.21: Transforms and coordinate frames involved in computing the distance to TSRs. The robot is in a sample configuration which has end-effector transform s, and the hand near the soda can at transform e represents the T_e^ω defined by the TSR [8]

5.3.2.1 TSR Definition

To define the TSR, transformation matrices, that were explained previously in this chapter, are used. Recalling, a transform $\mathbf{T}_b^a$ specifies the pose of b in the coordinates of frame a. $\mathbf{T}_b^a$ written in homogeneous coordinates consists of a 3×3 rotation matrix $\mathbf{R}_b^a$ and a 3×1 translation vector $\mathbf{t}_b^a$:

$$\mathbf{T}_b^a = \begin{bmatrix} \mathbf{R}_b^a & \mathbf{t}_b^a \\ \mathbf{0} & 1 \end{bmatrix}$$

A TSR consists of three parts [8]:

- $\mathbf{T}_\omega^o$: transform from the origin (task-space) to the TSR frame ω.
- $\mathbf{T}_e^\omega$: end-effector offset transform in the coordinates of ω.
- $\mathbf{B}^\omega$: 6×2 matrix of bounds in the coordinates of ω:

$$\mathbf{B}^{\omega} = \begin{bmatrix} x_{min} & x_{max} \\ y_{min} & y_{max} \\ z_{min} & z_{max} \\ \psi_{min} & \psi_{max} \\ \theta_{min} & \theta_{max} \\ \phi_{min} & \phi_{max} \end{bmatrix}$$

The first three rows of $\mathbf{B}^{\omega}$ bound the allowable translation along the $x-$, $y-$, and $z-$axes (in meters) and the last three bounds the allowable rotation around those axes (in radians), all in the ω frame. Note that this assumes the Roll-Pitch-Yaw (RPY) Euler angle convention, which is used because it allows bounds on rotation to be intuitively specified. Also, note that the end-effector frame (ω coordinates) is a robot-specific frame.

In practice, the ω frame is usually centered at the origin of an object held by the hand or at a location on an object that is useful for grasping. An end-effector offset transform $\mathbf{T}_e^{\omega}$ is used because it is not assumed that ω directly encodes the pose of the end-effector. This transform $\mathbf{T}_e^{\omega}$ allows the user to specify an offset from ω to the origin of the end-effector e, which is extremely useful when we wish to specify a TSR for an object held by the hand or a grasping location which is offset from e (see Figure 5.21) [8].

5.3.2.2 Distance to TSR

Sampling-based planning algorithms as TSR require a function that measures the distance between two points in $\mathcal{C}$, that is used to determine the adjacency of the configurations. Thus, it will be necessary to define the distance from a given configuration q_s to a TSR (Figure 5.21). Because there is not an analytical representation of the constraint manifold corresponding to a TSR, it is defined the distance in the task-space and, then, it is calculated in the frame end-effector reference frame (ω coordinates). Given a q_s, they are used forward kinematics to get the position of the end-effector at this configuration $\mathbf{T}_s^o$. Then, it is applied the inverse of the offset $\mathbf{T}_e^{\omega}$ to get $\mathbf{T}_{s'}^o$, which is the pose of the grasp location or the pose of the object held by the hand in task-space coordinates [7]:

$$\mathbf{T}_{s'}^{o} = \mathbf{T}_s^{o}(\mathbf{T}_e^{\omega})^{-1}$$

Then, this pose is converted from task-space coordinates to the coordinates of ω:

$$\mathbf{T}_{s'}^{\omega} = (\mathbf{T}_{\omega}^{o})^{-1}\mathbf{T}_{s'}^{o}$$

Now, it is converted the transform $\mathbf{T}_{s'}^{\omega}$ into a 6×1 displacement vector from the origin of the ω frame. This displacement represents rotation in the RPY convention so it is consistent with the definition of $\mathbf{B}^{\omega}$:

$$d^{\omega} = \begin{bmatrix} t_{s'}^{\omega} \\ \arctan 2\left(R_{s'_{32}}^{\omega}, R_{s'_{33}}^{\omega}\right) \\ -\arcsin\left(R_{s'_{31}}^{\omega}\right) \\ \arctan 2\left(R_{s'_{21}}^{\omega}, R_{s'_{11}}^{\omega}\right) \end{bmatrix}$$

Taking into account the bounds of $\mathbf{B}^{\omega}$, we get the 6×1 displacement vector to the TSR, $\triangle\mathbf{x}$:

$$\triangle\mathbf{x}_i = \begin{cases} d_i^{\omega} - \mathbf{B}_{i,1}^{\omega} & \text{if } d_i^{\omega} < \mathbf{B}_{i,1}^{\omega} \\ d_i^{\omega} - \mathbf{B}_{i,2}^{\omega} & \text{if } d_i^{\omega} > \mathbf{B}_{i,2}^{\omega} \\ 0 & \text{otherwise} \end{cases}$$

where i indexes through the six rows of $\mathbf{B}^{\omega}$ and six elements of $\triangle\mathbf{x}$ and d_{ω}.

5.3.2.3 Planning with TSRs

TSRs can be used to sample goal end-effector placements of a manipulator, as it would be necessary in a grasping or object-placement task. The constraint for using TSRs in this way is:

$$\{\mathcal{Q}(q) = \text{Distance}ToTSR(q), \;\; s = [1]\} \tag{5.3}$$

where s refers to the domain of the constraint.

To generate valid configurations in the $\mathcal{M}_Q$ manifold corresponding to this constraint, we can use direct sampling of TSRs and pass the sampled pose to an IK solver to generate a valid configuration. TSRs can also be used for planning with constraints on end-effector pose for the entire path. The constraint definition for such a use of TSRs differs from Eq. 5.3 in the domain of the constraint:

$$\{\mathcal{Q}(q) = \text{Distance}ToTSR(q), \;\; s = [0, 1]\} \tag{5.4}$$

Since the domain of this constraint spans the entire path, the planning algorithm must ensure that each configuration it deems valid lies within the constraint manifold [7].

The sampling-based planning requires an algorithm that searches globally while satisfying constraints locally. The sampling-based TSR planner is based on rapidly exploring random trees (RRT). The efficiency of these planners stems from the incomplete coverage of the free space and from terminating the search when the goal is first reached. The solution found is feasible but not optimal in any way [39]. These planners can also operate in the state space of the robot to produce trajectories, but they are far more efficient and successful when operating in the configuration-space to produce configuration-space paths. These paths can then be converted to trajectories and executed by an appropriate controller [8]. The implementation of TSR for the particular case of the HSR robot has already been done by the Toyota Research Institute (TRI).

Within the context of the TSR framework, the constraints can be easily represented and the planning of the end-effector achieved in the opening door task. When considering the set of objects relevant for a service robot, one quickly realizes that the joints in many articulated objects belong to a few generic classes. In particular, regarding the target objects of this project, the links could be divided as prismatic (drawer) or revolute (hinged door). Thus, in order to adress the motion planning problem, both cases should be written in the TSR formulation.

5.3.2.4 Opening a Drawer with TSR

For opening a drawer, the end-effector trajectory should be constrained in such a way that, once grasped, the handle follows a straight line in the direction perpendicular to the door plane. Thus, to specify the constraint in this problem only one TSR is defined, with T^o_ω to be coincident with the robot odometry. T^ω_e is the transform between the end-effector and the initial pose of the handle as defined in the previous chapter (Figure 5.22). In the same way, another TSR is defined for the goal:

$$(B^\omega_{constraint})^T = \begin{pmatrix} 0 & 0 & -d & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$(B^\omega_{goal})^T = \begin{pmatrix} 0 & 0 & -d & 0 & 0 & 0 \\ 0 & 0 & -d & 0 & 0 & 0 \end{pmatrix}$$

where d is the desired opening distance.

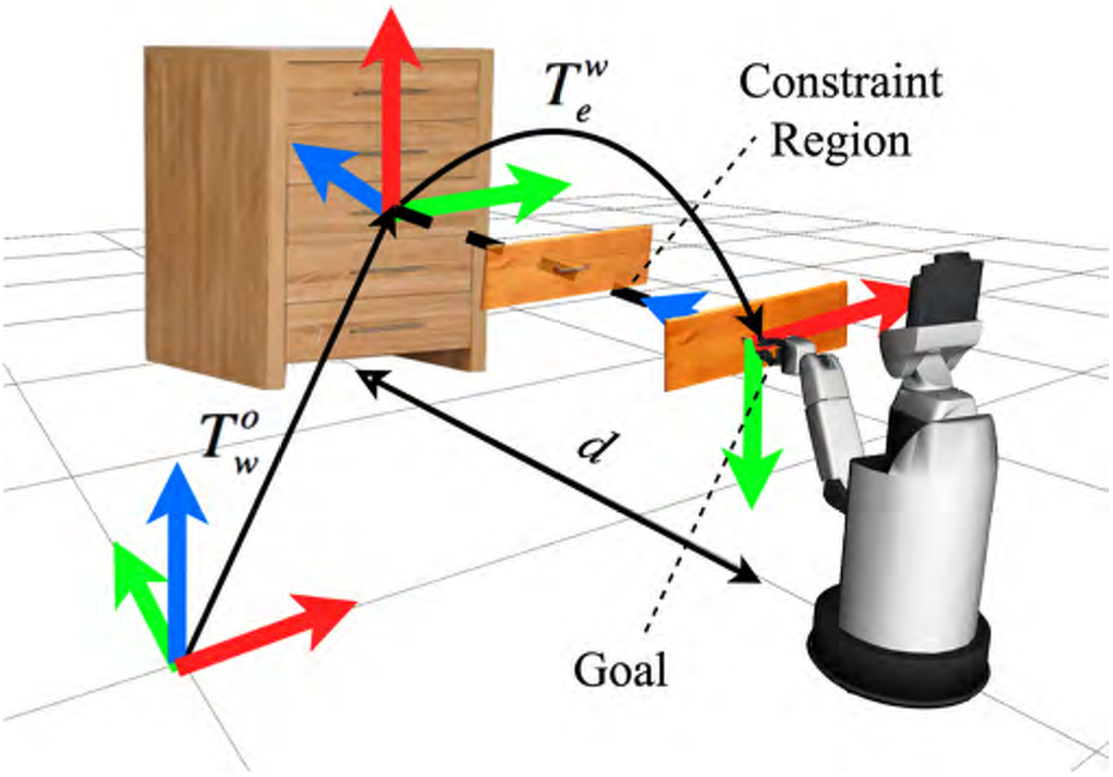


Figure 5.22: TSR planning to open a drawer

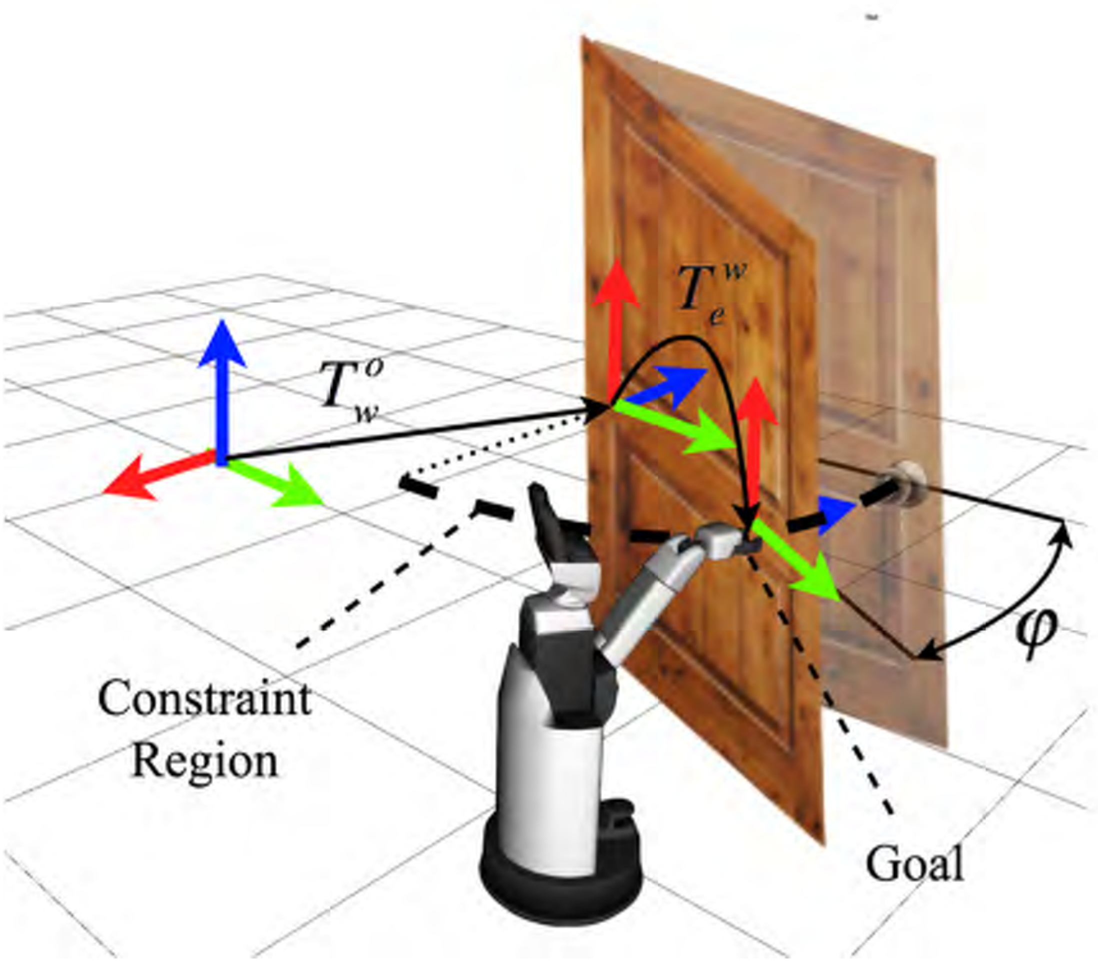


Figure 5.23: TSR planning to open a hinged door

5.3.2.5 Opening a Hinged Door with TSR

In this problem the task is to open a hinged door a certain angle φ. The planner is only allowed to move the door handle describing a circumference whose center has the position of the hinge and it is contained in a plane parallel to the floor. To specify the pose constraint in this problem, one pose constraint is defined with T_ω^o at the door hinge, with no rotation relative to the initial handle orientation (as it was defined in the previous chapter). The constraints and the goal in this particular case will depend on T_e^ω, since the sense of the rotation will be the opposite if its y- translation component is positive or negative, i.e., if the door handle is to the right or to the left from the axis of rotation.

Thus, the goal and constraint are (Figure 5.23):

$$\left.\begin{aligned}(B^{\omega}_{constraint})^T &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \varphi & 0 & 0 \end{pmatrix} \\ (B^{\omega}_{goal})^T &= \begin{pmatrix} 0 & 0 & 0 & \varphi & 0 & 0 \\ 0 & 0 & 0 & \varphi & 0 & 0 \end{pmatrix}\end{aligned}\right\} T^{\omega}_{e} \mid_y < 0$$

$$\left.\begin{aligned}(B^{\omega}_{constraint})^T &= \begin{pmatrix} 0 & 0 & 0 & -\varphi & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\ (B^{\omega}_{goal})^T &= \begin{pmatrix} 0 & 0 & 0 & -\varphi & 0 & 0 \\ 0 & 0 & 0 & -\varphi & 0 & 0 \end{pmatrix}\end{aligned}\right\} T^{\omega}_{e} \mid_y > 0$$

6 Learning

Enabling robots to act autonomously in the real-world is difficult.Why are autonomous robots not out in the world among people? Robots are particularly susceptible to Murphy's law: everything that can go wrong, will go wrong. Robots struggle in unknown scenarios, and it is very difficult for them to deal with uncertainty. However, either do humans. Where is the difference then? That humans can come out of these situations, because they use their own experience to overcome uncertainty, but also, because they learn. If the human being the only animal that trips twice over the same stone, robots will trip a dozen times. Two important concepts have arised so far in this discussion: uncertainty and learning. Uncertainty is a situation which involves imperfect or unknown information. It applies to predictions of future events, to physical measurements that are already made, or to the unknown. Learning is the activity of gaining knowledge or skill by studying, practising, being taught, or experiencing something. In order to make reliable robotic systems operating in human enviroments both have to be adressed.

Regarding uncertainty, the following question could be made: What if a robot has access to sensors that allow it to observe the world state, but no previous informative knowledge on it at the time of decision? Bayes in 1763 and Laplace later in 1812 already answered this question. If there is no prior knowledge but observations are available, the best decision will be the one that has the highest chances of being correct given the observations [23]. Regarding the learning process, in particular learning from experience, the previous question can be reformulated: What if a robot has access to sensors that allow it to observe the world state, and also previous informative knowledge on it at the time of decision? Bayes and Laplace already solved this question too! The correct decision will be the one that has the highest probability of being correct balancing the given observations and the prior knowledge (Figure 6.1).

This is Bayesian inference, a method of statistical inference in which Bayes theorem is used to update the probability for a hypothesis as more evidence or information becomes available. Bayesian inference derives the posterior probability as a consequence of two antecedents: a prior probability and a "likelihood function" derived from a statistical model for the observed data. The challenge for human assistive robots is that these robots, that operate in unstructured environments, have to cope with less prior knowledge about their surroundings. Therefore, they need to be able to autonomously learn suitable probabilistic models from their own sensor data to robustly fulfill their tasks.

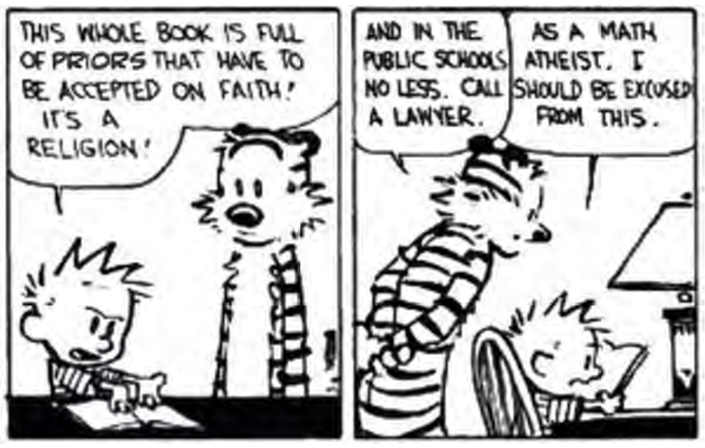

THIS WHOLE BOOK IS FULL OF PRIORS THAT HAVE TO BE ACCEPTED ON FAITH! IT'S A RELIGION!
AND IN THE PUBLIC SCHOOLS NO LESS. CALL A LAWYER.
AS A MATH ATHEIST, I SHOULD BE EXCUSED FROM THIS.

Figure 6.1: Prior knowledge is the basis of the learning, even though Calvin and Hobbes disagree to some extent

In this chapter, a probabilistic framework to learn the doors kinematic model, take advantage of previous experiences to improve the performance, proposed by Sturm et al. [64, 74–76], will be devoloped and extended. In section 6.1, some statistical insights of the proposed approach will be presented. Then, in section 6.2, a Bayesian framework to learn the kinematic model of doors will be developed. In section 6.3, the candidate models for parametrizing the door motion will be described. Next, in section 6.4, an approach to fit the observations into the previous models will be presented. In section 6.5, it will be described the procedure to select the most probable model. Then, in section 6.6, it will be proposed an approach to integrate the probabilistic framework in a robot control scheme to achieve the correct operation of unknwn doors. In section 6.7, it will be described how can the prior knowledge from previous experience be exploited to boost the robot performance. Next, in section 6.8, an approach that allows the robot to learn from human demonstrations will be proposed. And, finally, in section 6.9, an extension of the framework for building a semantic map will be described.

6.1 Model Based Learning

Machine learning offers to robotics a framework and set of tools for the design of sophisticated and hard-to-engineer behaviors; conversely, the challenges of robotic problems provide both inspiration, impact, and validation for developments in robot learning. In recent years, methods to learn models from data have become increasingly interesting tools for robotics.

A robot is learning if it improves its performance on future tasks after making observations about the world. For this, the robot has to consider two major issues. First, it needs to deduce the behavior of the system from some observed quantities. Second, having inferred this information, it needs to determine how to manipulate the system.

The first question is a pure modeling problem. Given some observed quantities, we need to predict the missing information to complete our knowledge about the action and system reaction. Depending on what kind of quantities are observed, the agent builds some kind of model to act on the system. The second question is related to the learning control architectures which can be employed in combination with these models. This is known as a model learning approach [17]. In general, model learning is a supervised learning approach. In supervised learning, the agent observes some example input–output pairs and learns a function that maps from input to output. Thus, approximating this underlying function is the goal of supervised learning methods. Given known input data, the learned model should be able to provide precise predictions of the output values [52]. The dependency between the observed data and the model is defined by the conditional probability distribution [75]:

$$p\left(\mathbf{y} \mid \mathbf{x}, \mathcal{M}\right)$$

which refers to the probability distribution of the random (or target) variable $\mathbf{y}$ given the value of the input variable $\mathbf{x}$ and a model $\mathcal{M}$.

If a deterministic relationship between input and target space exists, the model can be specified using a regression function that defines the functional mapping $f_{\mathcal{M}}$ from input to target space, i.e.,

$$\mathbf{y} = f_{\mathcal{M}}(\mathbf{x})$$

The goal of regression is to estimate $f_{\mathcal{M}}(\mathbf{x})$ from a set of n observations $D = \{(x_i,\, y_i)\}_{i=1}^{n}$. A straightforward way to define the model $\mathcal{M}$ is by using a parametric structure. The parametric approach to regression is to express the unknown regression function with a function $f_{\mathcal{M},\theta}$ that is parametrized by a vector. The aim is, then, to select the parameter vector $\boldsymbol{\theta}$ that best fits the data, or equivalently, that maximizes the posterior probability after having observed the dataset $\mathcal{D}$, i.e.,

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} p\left(\boldsymbol{\theta} \mid \mathcal{D}, \mathcal{M}\right)$$

Applying Bayes rule this is equivalent to:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \frac{p\left(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{M}\right) p\left(\boldsymbol{\theta} \mid \mathcal{M}\right)}{p\left(\mathcal{D}\right)} = \arg\max_{\boldsymbol{\theta}} p\left(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{M}\right) p\left(\boldsymbol{\theta} \mid \mathcal{M}\right)$$

where $p\left(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{M}\right)$ is called the data likelihood and $p\left(\boldsymbol{\theta} \mid \mathcal{M}\right)$ is the prior over the parameter space. The prior probability of the observed data $p(\mathcal{D})$ is neglected as it is independent of the choice of the parameter vector $\boldsymbol{\theta}$.

The model $\mathcal{M}$ is selected from a hypothesis space, $\mathcal{H}$. A fundamental problem is how a decision can be made from among the multiple consistent hypothesis. In general, there is a tradeoff between complex hypotheses that fit the training data well and simpler hypotheses that may generalize better. Selecting the most-likey model requires the evaluation and comparison of the posterior probabilities $p\left(\mathcal{M} \mid \mathcal{D}\right)$ of all models, i.e., [65].

$$\hat{\mathcal{M}} = \arg\max_{\mathcal{M} \in \mathcal{H}} p\left(\mathcal{M} \mid \mathcal{D}\right)$$

By Bayes rule this is equivalent to:

$$\hat{\mathcal{M}} = \arg\max_{\mathcal{M}\in\mathcal{H}} \frac{p\left(\mathcal{D} \mid \mathcal{M}\right) \cdot p\left(\mathcal{M}\right)}{p(\mathcal{D})} = \arg\max_{\mathcal{M}\in\mathcal{H}} p\left(\mathcal{D} \mid \mathcal{M}\right) \cdot p\left(\mathcal{M}\right)$$

where $p(\mathcal{M})$ is the prior over the models, i.e., the probability of a hypothesis model $\mathcal{M}$ to be the true function before having any data; and $p(\mathcal{D} \mid \mathcal{M})$ is the data likelihood. The prior probability of the observed data $p(\mathcal{D})$ is neglected as it is independent of the choice of the model $\mathcal{M}$.

6.2 Learning the Kinematic Models of Doors

Doors are articulated objects that consist of two rigid parts with a mechanical link. This link constrain the motion between the parts: for example, the rails of a drawer constrain it to move on a line and the hinge of a door constrains the door to move on an arc. The problems that have to be addressed at this point to enable the robot to open the door are: the inference of the kinematic model and its corresponding parametrization. In this way, a motion planning to finally open the door could be achieved using the TSR framework presented in the previous chapter. By means of the probabilistic approach presented in the previous section, these unknowns can be solved under some assumptions.

The robot observes a sequence of N relative transformations $\mathcal{D} = (\mathbf{d}_1, ..., \mathbf{d}_N)$ between two adjacent rigid parts of this object. Our observational model assumes that all the measurements are affected by Gaussian noise. Furthermore, a small fraction of this observations are outliers that cannot be explained by the Gaussian noise assumption alone. These outliers may be the result of poor perception, bad data association, or other sensor failures that are hard to be modeled explicitly. The kinematic link model is denoted as $\mathcal{M}$ and its associated parameter vector as $\boldsymbol{\theta} = \mathbb{R}^k$ (where k corresponds to the number of the parameters describing the model). The model that best represents the data, and its corresponding parameters, can be computed as follows:

$$(\hat{\mathcal{M}}, \hat{\boldsymbol{\theta}}) = \arg\max_{\mathcal{M},\boldsymbol{\theta}} p(\mathcal{M}, \boldsymbol{\theta} \mid \mathcal{D})$$

Solving this equation is a two-step process [46]:

- At the first level of inference, a particular model is assumed true and its parameters are estimated from the observations.

  $$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \mathcal{D}, \mathcal{M})$$

  By applying Bayes rule, this can be rewritten into:

  $$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \frac{p(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{M}) p\left(\boldsymbol{\theta} \mid \mathcal{M}\right)}{p\left(\mathcal{D} \mid \mathcal{M}\right)} = \arg\max_{\boldsymbol{\theta}} p(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{M}) p\left(\boldsymbol{\theta} \mid \mathcal{M}\right)$$

where the term $p(\boldsymbol{\theta} \mid \mathcal{M})$ defines the model-dependent prior over the parameter space. If the prior is assummed to be uniform, it follows:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} p(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{M}) \tag{6.1}$$

which means that a fitting of a link model to the observations corresponds to the problem of maximizing data likelihood.

- At the second level of inference, the probability of different models is compared, and the model with the highest posterior probability is selected:

$$\hat{\mathcal{M}} = \arg\max_{\mathcal{M}} p\,(\mathcal{M} \mid \mathcal{D}) = \arg\max_{\mathcal{M}} \int p\,(\mathcal{M}, \boldsymbol{\theta} \mid \mathcal{D})\, d\boldsymbol{\theta} \tag{6.2}$$

Therefore, from observations of the door motion, the robot could be able:

1. Fit the parameters of all candidate kinematic models that describe the different mechanical links.
2. Select the kinematic model that best explains the observed motion, i.e., the kinematic structure that maximizes the posterior probability.

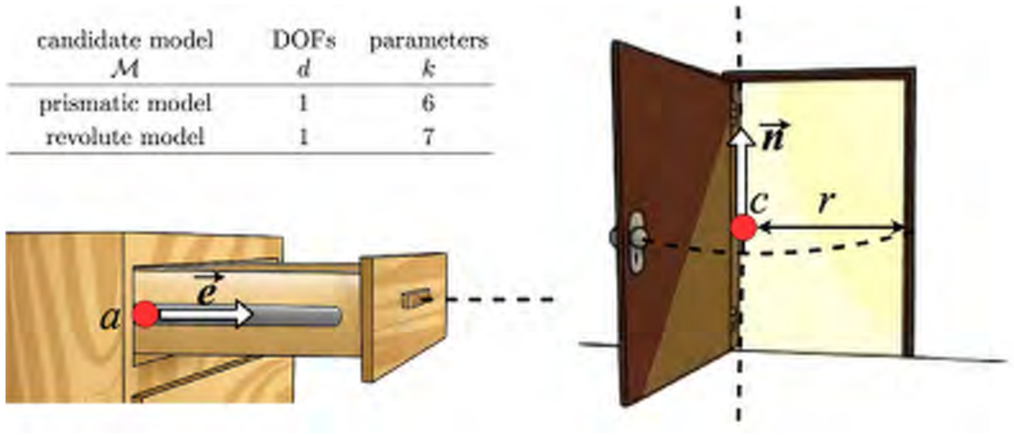
6.3 Candidate Models

Kinematic models describe the motion of systems of articulated objects such as doors. When considering the set of doors that can be potentially operated by a service robot, the joints belong to a few generic classes, in particular, prismatic (drawers) and revolute joints (hinged doors). Thus, the candidate set of models $\mathcal{H}$ includes a prismatic model $\mathcal{M}_{prism}$ and a revolute model $\mathcal{M}_{rev}$.

These models have a latent variable $\mathbf{q} \in \mathbb{R}^d$ that describes the configuration of the mechanism, where d represents the number of the DOFs of the mechanical link. For both prismatic and revolute models, $d = 1$, so the parameter $q \in \mathbb{R}$ is a scalar.

Prismatic model Prismatic joints move along a single axis, and thus have a one-dimensional configuration space (Figure 6.2). The prismatic model describes a translation along a vector of unit length $\mathbf{e} \in \mathbb{R}^3$ relative to some fixed origin, $\mathbf{a} \in \mathbb{R}^3$. This results in a parameter vector $\boldsymbol{\theta} = (\mathbf{a}; \mathbf{e})$ with $k = 6$ dimensions. Thus, only the coordinates of two points are needed to define it.

Revolute model Revolute joints rotate around an axis that impose a one-dimensional motion along a circular arc (Figure 6.2). This model is parametrized by the center of rotation $\mathbf{c} \in \mathbb{R}^3$, a radius $r \in \mathbb{R}$, and the normal vector $\mathbf{n} = \mathbb{R}^3$ to the plane where the motion is contained. This results in a parameter vector $\boldsymbol{\theta} = (\mathbf{c}; \mathbf{n}; r)$ with $k = 7$ dimensions. Thus, the statistical adjustment of the model only requires 7 parameters.



candidate model $\mathcal{M}$	DOFs d	parameters k
prismatic model	1	6
revolute model	1	7

Figure 6.2: Prismatic and revolute models for articulated links

6.4 Model Fitting

For estimating the parameters of any of the above-mentioned models, the parameter vector $\hat{\boldsymbol{\theta}} \in \mathbb{R}^k$ that maximizes the data likelihood given the model:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} p\left(\mathcal{D} \mid \mathcal{M}, \boldsymbol{\theta}\right)$$

In the presence of noise and outliers, finding the right parameter vector $\boldsymbol{\theta}$ that maximizes the data likelihood is not trivial, as least squares estimation is sensitive to outliers and thus not sufficient given our observation model. RANSAC algorithm, explained in section 4.2.2, has proven to be very successful for robust estimation. An improvement can be obtained modifying RANSAC in order to maximize the likelihood of the model. This is the approach implemented by MLESAC (Maximum Likelihood Estimation SAmple and Consensus), a variation of RANSAC that evaluates the quality of the consensus set (i.e., the data that fit a model and a certain set of parameters) calculating its likelihood (whereas in RANSAC formulation the rank was the cardinality of such set). The main idea of MLESAC is to evaluate the likelihood of the hypothesis by representing the error distribution as a mixture model of a Gaussian (inliers) and a uniform (outliers) distributions (Figure 6.3). The MLESAC algorithm adjusts the vector parameter $\boldsymbol{\theta}$ of a model $\mathcal{M}$ works as follows:

- First, a guess for the parameter vector $\hat{\boldsymbol{\theta}}$ is generated from a minimal set of samples from the whole observation sequence $\mathcal{D}$ (like in the RANSAC algorithm).
- For this guess, the data likelihood of the observation sequence (N observations: $\mathbf{d}_1, ..., \mathbf{d}_N$) can be expressed as [78, 87]:

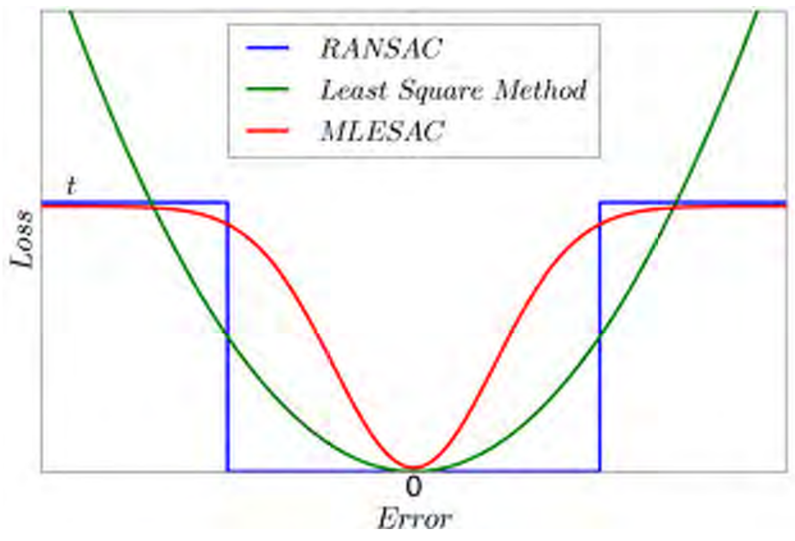


Figure 6.3: RANSAC, LSE, and MLESAC loss functions [14]

$$p(\mathcal{D} \mid \mathcal{M}, \hat{\boldsymbol{\theta}}) = \prod_{i=1}^{N} p\left(\mathbf{d}_i \mid \mathcal{M}, \hat{\boldsymbol{\theta}}\right) = \prod_{i=1}^{N} \Big(\gamma \cdot p\left[e(\mathbf{d}_i, \mathcal{M}, \hat{\boldsymbol{\theta}}) \mid \text{the i}^{\text{th}} \text{ element is an inlier}\right] +$$

$$(1-\gamma \cdot)\, p\left[e(\mathbf{d}_i, \mathcal{M}, \hat{\boldsymbol{\theta}}) \mid \text{the i}^{\text{th}} \text{ element is an outlier}\right]\Big) \tag{6.3}$$

where γ is the outlier ratio or the mixture coefficient.

- The error distribution for the inliers is modeled with a Gaussian distribution:

$$p_i = p\left[e(\mathbf{d}_i, \mathcal{M}, \boldsymbol{\theta}) \mid \text{the i}^{\text{th}} \text{ element is an inlier}\right] = \frac{1}{Z} \exp\left[\frac{-e(\mathbf{d}_i, \mathcal{M}, \boldsymbol{\theta})^2}{2\Sigma_{\mathbf{d}}^2}\right] \tag{6.4}$$

where Z is the appropiate normalization constant, and $\Sigma_{\mathbf{d}}$ indicates the noise standard deviation.

- The error statistics for the outliers is described by a uniform distribution:

$$p_o = p\left[e(\mathbf{d}_i, \mathcal{M}, \boldsymbol{\theta}) \mid \text{the i}^{\text{th}} \text{ element is an outlier}\right] = \begin{cases} \frac{1}{2e_{max}} & \mid e(\mathbf{d}_i, \mathcal{M}, \boldsymbol{\theta}) \mid \leq e_{max} \\ 0 & \text{otherwise} \end{cases} \tag{6.5}$$

where e_{max} represents the largest possible error which can be induced by the presence of outliers.

- Note that two parameters need to be estimated, the vector parameter $\hat{\boldsymbol{\theta}}$ that maximizes the likelihood of data observations, and the mixture coefficient:

– The γ estimation can be done iterativily using Expectation Maximization (EM), for which a set of indicator variables needs to be introduced: η_i $(i = 1, ...n)$, where $\eta_i = 1$ if the $i-$th datum is an inlier, and $\eta_i = 0$ if the $i-$th datum is an outlier. The algorithm EM proceeds as follows, treating the η_i as missing data:

1. Generate a guess for γ (usually the initial estimate of γ is $\frac{1}{2}$);
2. Estimate the expectation of the η_i from the current estimate of γ. Here, $p(\eta_i = 1 \mid \gamma) = z_i$ and, given an value of γ, this can be estimated as:

$$z_i = p(\eta_i = 1 \mid \gamma) = \frac{p_i}{p_i + p_o}$$

then, $p(\eta_i = 0 \mid \gamma) = 1 - z_i$, where, p_i is the likelihood of a datum is an inlier (Eq. 6.4), and p_o is the likelihood of a datum is an outlier (Eq. 6.5).

3. Make a new estimate of γ from the current estimate of η_i and go to step (2):

$$\gamma = \frac{1}{n} \sum_i z_i$$

4. Repeat this procedure until convergence (typically it requires two or three iterations).

– For estimating the vector parameter $\hat{\boldsymbol{\theta}}$, the log-likelihood of the mixture model is maximized:

$$\hat{\mathcal{L}} \left[e(\mathcal{D} \mid \mathcal{M}, \hat{\boldsymbol{\theta}}) \right] = \sum_{i=1}^{N} \log \Big(\gamma \cdot p \left[e(\mathbf{d}_i, \mathcal{M}, \hat{\boldsymbol{\theta}}) \mid \text{the i}^{\text{th}} \text{ element is an inlier} \right] +$$

$$(1 - \gamma \cdot)\, p \left[e(\mathbf{d}_i, \mathcal{M}, \hat{\boldsymbol{\theta}}) \mid \text{the i}^{\text{th}} \text{ element is an outlier} \right] \Big)$$

These steps are repeated for a fixed number of iterations and finally the parameter vector maximizing Equation 6.3 are selected. In this way, the parameters and the data likelihood are computed. This algoritm is applied for each one of the candidate models proposed in the previous section.

Then, the question of how to decide which of the two models is more likely to represent the data according to Equation 6.2 arises. This will be adressed in the next section.

6.5 Model Selection

After having fitted all model candidates to an observation sequence $\mathcal{D}$, we need to select the model that explains the data best has to be selected. For Bayesian model selection, this means that the posterior probability of the models (given the data) needs to be compared:

$$p(\mathcal{M} \mid \mathcal{D}) = \int \frac{p\left(\mathcal{D} \mid \mathcal{M}, \boldsymbol{\theta}\right) p\left(\boldsymbol{\theta} \mid \mathcal{M}\right) p\left(\mathcal{M}\right)}{p\left(\mathcal{D}\right)} d\boldsymbol{\theta}$$

Let $\mathcal{M}_m$ $(m = 1, ..., M)$ be the set of candidate models, and their corresponding model parameters $\boldsymbol{\theta}_m$. Let $p\left(\boldsymbol{\theta}_m \mid \mathcal{M}_m\right)$ be the prior distribution for the parameters of each model $\mathcal{M}_m$. Thus, the posterior probability of a given model is [27]:

$$p\left(\mathcal{M}_m \mid \mathcal{D}\right) = \frac{p(\mathcal{M}_m) \cdot p\left(\mathcal{D} \mid \mathcal{M}_m\right)}{p(\mathcal{D})} = \frac{p\left(\mathcal{M}_m\right)}{p(\mathcal{D})} \cdot \int p\left(\mathcal{D} \mid \boldsymbol{\theta}_m, \mathcal{M}_m\right) p\left(\boldsymbol{\theta}_m \mid \mathcal{M}_m\right) d\boldsymbol{\theta}_m \qquad (6.6)$$

where $\mathcal{D}$ represents the training data $\{x_i, y_i\}_1^N$, (and N is the number of training samples).

In general, the evaluation of the model posterior probability is difficult, but the comparison between two models M_m and $M_{m'}$ can be solved by the posterior probability ratio:

$$\frac{p\left(\mathcal{M}_m \mid \mathcal{D}\right)}{p\left(\mathcal{M}_{m'} \mid \mathcal{D}\right)} = \frac{p\left(\mathcal{M}_m\right)}{p\left(\mathcal{M}_{m'}\right)} \cdot \frac{p\left(\mathcal{D} \mid \mathcal{M}_m\right)}{p\left(\mathcal{D} \mid \mathcal{M}_{m'}\right)}$$

The quantity:

$$BF(\mathcal{D}) = \frac{p\left(\mathcal{D} \mid \mathcal{M}_m\right)}{p\left(\mathcal{D} \mid \mathcal{M}_{m'}\right)}$$

is called the Bayes factor, and represents the contribution of the data toward the posterior probability ratio. If the prior over models is assumed uniform, so that $p(\mathcal{M}_m)/p(\mathcal{M}_{m'}) = 1$, the best model should be M_m while $BF(\mathcal{D}) > 1$. Then, in order to compare the models, it is necessary to approximate in some way $p\left(\mathcal{D} \mid \mathcal{M}_m\right)$. From Eq. 6.6:

$$p\left(\mathcal{D} \mid \mathcal{M}\right) = \int p\left(\mathcal{D} \mid \boldsymbol{\theta}_m, \mathcal{M}_m\right) p\left(\boldsymbol{\theta}_m \mid \mathcal{M}_m\right) d\boldsymbol{\theta}_m$$

Using the Laplace approximation to the integral, followed by some other simplifications, it follows:

$$\log p\left(\mathcal{D} \mid \mathcal{M}_m\right) = \log p\left(\mathcal{D} \mid \hat{\boldsymbol{\theta}}_m, \mathcal{M}_m\right) - \frac{k_m}{2} \cdot \log N + c \qquad (6.7)$$

where $\hat{\boldsymbol{\theta}}_m$ is a maximum likelihood estimate, k_m is the number of free parameters in model $\mathcal{M}_m$, and c is a constant.

Equation 6.7 is very similar to the Bayesian Information Criteria (BIC) used in statistics. The BIC arises in a Bayesian approach to model selection, and it is defined as [10]:

$$BIC = -2\log(\hat{\mathcal{L}}) + k \cdot \log N = -2\log\left[\mathcal{L}\left(\mathcal{D} \mid \mathcal{M}, \hat{\boldsymbol{\theta}}\right)\right] + k \cdot \log N \tag{6.8}$$

where $\hat{\mathcal{L}}$ is the maximum value of the likelihood function for the model; $\mathcal{D}$ is the observation sequence; $\hat{\boldsymbol{\theta}}$ is the maximum likelihood parameter vector of the model (Eq. 6.2); k is the number of parameters of the current model under consideration, $\mathcal{M}$; and N is the number of observations in the sequence $\mathcal{D}$. The BIC incorporates two terms (Eq. 6.8): the first of them improves the quality of the adjustment (higher likelihood) and the second penalizes the complexity of the model [1].

Then, model selection is reduced to select the model that has the lowest BIC, i.e.:

$$\hat{\mathcal{M}} = \arg\min_{\mathcal{M}} BIC(\mathcal{M})$$

The individual BIC values are not interpretable, as they contain arbitrary constants and are much affected by sample size. Here, it is convenient to rescale BIC values to $\triangle BIC_i = BIC_i - BIC_{min}$, where BIC_{min} is the minimum of the different BIC values of the considered models. This transformation forces the best model to have $\triangle BIC = 0$, while the rest of the models have positive values. These $\triangle BIC$ values allow meaningfull interpretation without the unknow scaling constants and sample size issues that enter into BIC values [10]. If the BIC is computed for a set of models $\{\mathcal{M}_i\}_1^M$, the posterior probability of each model $\mathcal{M}_i$ as [27]:

$$p_i = p(\mathcal{M}_i \mid D) = \frac{\exp\left(-\frac{1}{2}\triangle BIC_i\right)}{\sum_{i=1}^{M}\left(-\frac{1}{2}\triangle BIC_i\right)}$$

The above posterior model probabilities are based on assuming that prior model probabilities are all the same, that is $1/M$. Thus, we can estimate not only the best model, but also can be estimated the relative merits of the considered models. In the following section it will be presented an approach to exploit this framework in order to open unknown doors with the Toyota HSR.

6.6 Opening Unknown Doors

The only remaining piece in the proposed approach is how can the robot collect observations and use the probablilistic framework to robustly operate unknown doors. When the robot establishes firm contact with the handle of the door, the position of its end effector directly corresponds to the position of the door handle. As a result, the robot can both sense the position of the handle as well as control it by moving the manipulator. Then, the robot observes the position of its end effector in Cartesian space, denoted by $\mathbf{d} \in \mathbb{R}^3$. While operating the object, the robot records the trajectory $\mathbf{d}_i(t)$ over time as a sequence of positions. To drive the robot actuation, the estimated model parameters are used within the TSR framework as it was presented in section 5.3.2 (Figure 6.4).

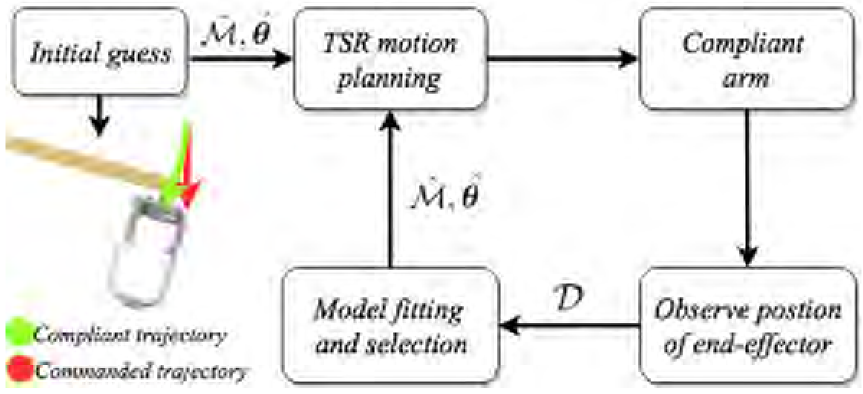

Initial guess
$\hat{\mathcal{M}}, \hat{\theta}$
TSR motion planning
Compliant arm
$\hat{\mathcal{M}}, \hat{\theta}$
Model fitting and selection
$\mathcal{D}$
Observe postion of end-effector
Compliant trajectory
Commanded trajectory

Figure 6.4: Door opening motion: each step corresponds to one loop in the diagram

The door opening motion is divided in steps. After each step, the kinematic model of the door and its parameters are re-estimated adding the new observations, since the larger the set the more accurate will be the estimation. For driving the robot motion in accordance with the model, the Task Space Region (TSR) framework is used. For starting the manipulation process, where no observations are acquired, a prismatic model is supposed, since small portions of revolute trajectories can be approximated by straight lines. Compliance plays a key role. Impedance control is used in order to make the robot arm compliant with the door. In this way, the robot is able to open the door correctly even if the model selection is wrong at the beginning, since the end-effector trajectory is also controlled by the forces exerted from the door. This allows an error margin in the estimated kynematic model which is fundamental. Once there are enough observations, the estimation is more accurate and this error margin is no longer needed.

6.6.1 Results

The performance of the proposed approach was tested ten times with three different types of doors. These were a drawer, a hinged door and a refrigerator door. The task of the robot was to grasp the handle, unlock it and open the door while it learned its kinematic model. The robot succesfully opened the doors 26 times out of the 30 trials (87%). All four failures were due to the robot gripper slipping from the door knob, most likely because of the design of the gripper is not very suitable to operate these kind of objects. No errors were observed during the model learning. Figure 6.5 shows the HSR opening the three types of doors. In a second series of experiments, the convergence behavior of the estimators with respect number of training samples was evaluated. Both a door with a prismatic model (drawer) and another with a revolute model (refrigerator door) were opened succesfully ten times each one. During the task, the evolution of the candidate model posterior was tested against the number of observations as well as, in the case of the revolute model, the error in the radius estimation. As it can be observed in Figure 6.6 and Figure 6.7, the posterior probability of both models converges towards the true model as the number of observations increases.

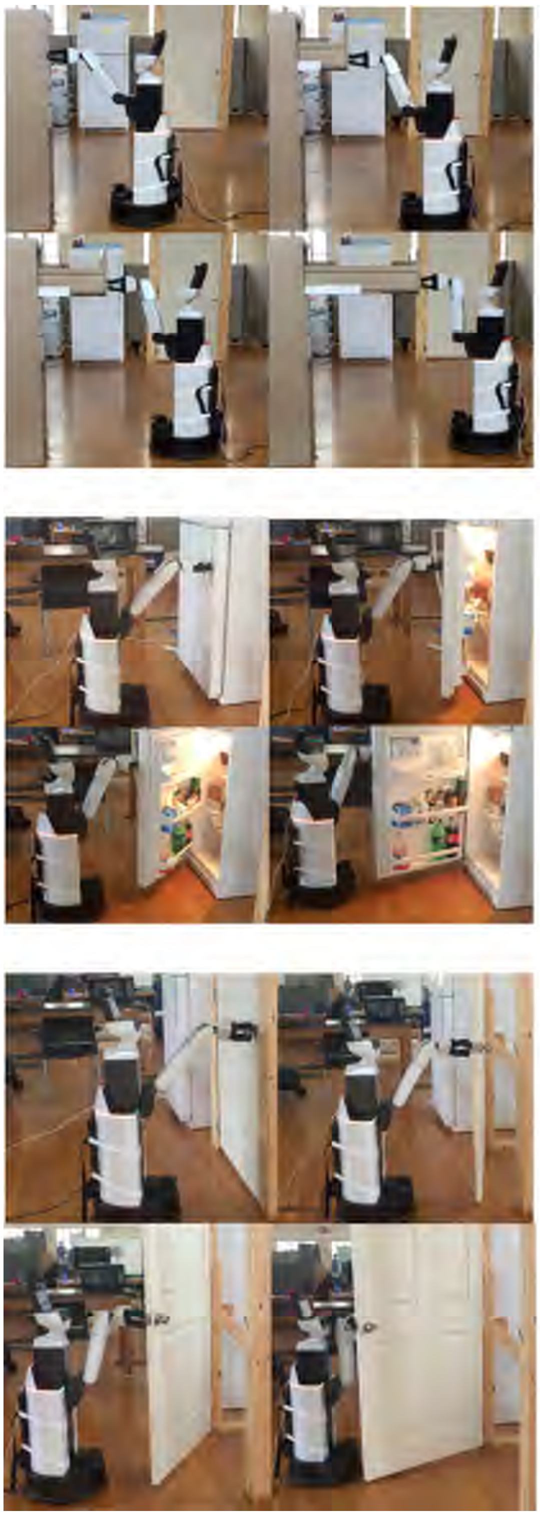


Figure 6.5: HSR opening, from top bottom: a drawer, a refrigerator door, and a room door

When few observations are acquired the probability oscillates around 0.5, which is consistent with the hypothesis of considering equal priors for both candidate models. However, they soon diverge from this value, showing an efficient behavior regarding the decision criteria. A more convergent behavior is appreciated in the case of the opening of a revolute door. This is the case because of the difference in complexity between both models. When fitting a revolute model into a prismatic trajectory, the like-lihood is still meaningful since a circumference with a big radius can approximate the mentioned trajectory. However, when fitting a prismatic model into a revolute trajectory this is not the case. On the other hand, a convergent behavior in the radius estimation when opening a revolute door, rapidly decreasing to aceptable values to gurarantee a satisfactory operation of the robot can also be observed.

These results show that, with the proposed approach, the robot can learn the kinematic models of unknown doors with a robust behavior. Also, the probabilistic framework is applicable to a wide range of doors, allowing the robot to operate them reliably.

6.7 Learning from Experience: Exploiting Prior Knowledge

So far, with the presented framework, a robot always learns the model from scratch when it opens a door. However, doors in indoor enviroments present very similar kynematic models. Thus, a robot operating in such environments over extended periods of time can significantly boost its performance by learning priors over the space of possible articulated object models. Then, the proposed framework can be extended to allow the robot learn priors for doors and exploit them as early as possible while manipulating unknown doors. Thus, the objective is to transfer the information contained in already learned models to newly seen doors. The key idea here is to identify a small set of representative models for the articulated objects and to utilize this as prior information to improve the model selection and parameter estimation [75].

Let's suppose that the robot has previously encountered two doors. Their observed motion is given by two observation sequences $\mathcal{D}_1$ and $\mathcal{D}_2$, with n_1 and n_2 sample respectively. The question now is whether both trajectories should be described by two distinct models $\mathcal{M}_1$ and $\mathcal{M}_2$ or by a joint model $\mathcal{M}_{1+2}$. In the first case, the posterior can be split as the two models are mutually independent, i.e. [76]:

$$p(\mathcal{M}_1, \mathcal{M}_2 \mid \mathcal{D}_1, \mathcal{D}_2) = p(\mathcal{M}_1 \mid \mathcal{D}_1) \cdot p(\mathcal{M}_2 \mid \mathcal{D}_2) \tag{6.9}$$

In the latter case, both trajectories are explained by a single, joint model $\mathcal{M}_{1+2}$ with a parameter vector $\boldsymbol{\theta}_{1+2}$, that is estimated from the joint data $\mathcal{D}_1 \cup \mathcal{D}_2$. The corresponding posterior probability of this joint model is denoted as:

$$p(\mathcal{M}_{1+2} \mid \mathcal{D}_1, \mathcal{D}_2) \tag{6.10}$$

To determine whether a joint model is better than two separate models by comparing the posterior probabilities from Eq. 6.9 and Eq. 6.10, i.e., by evaluating:

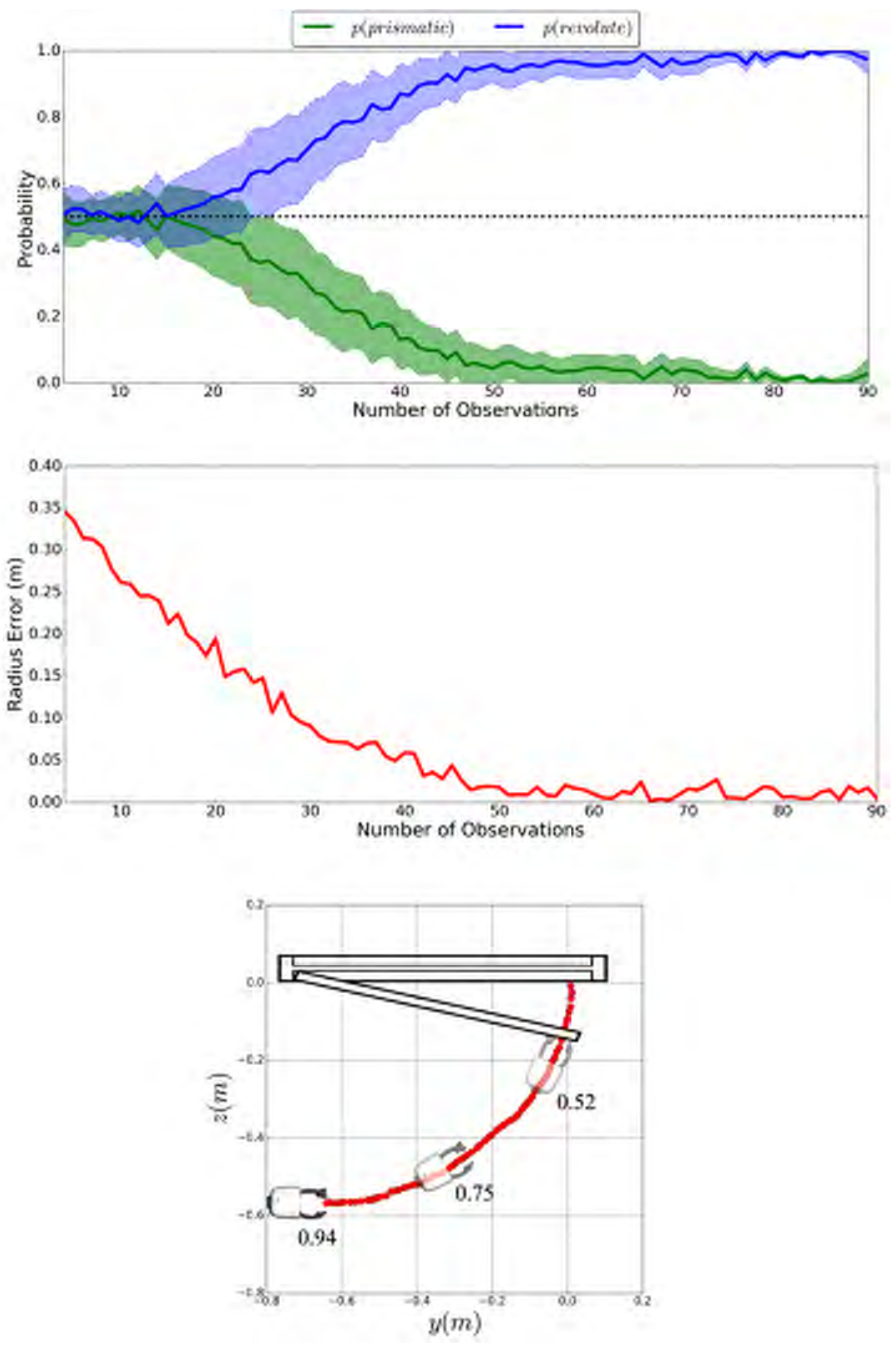
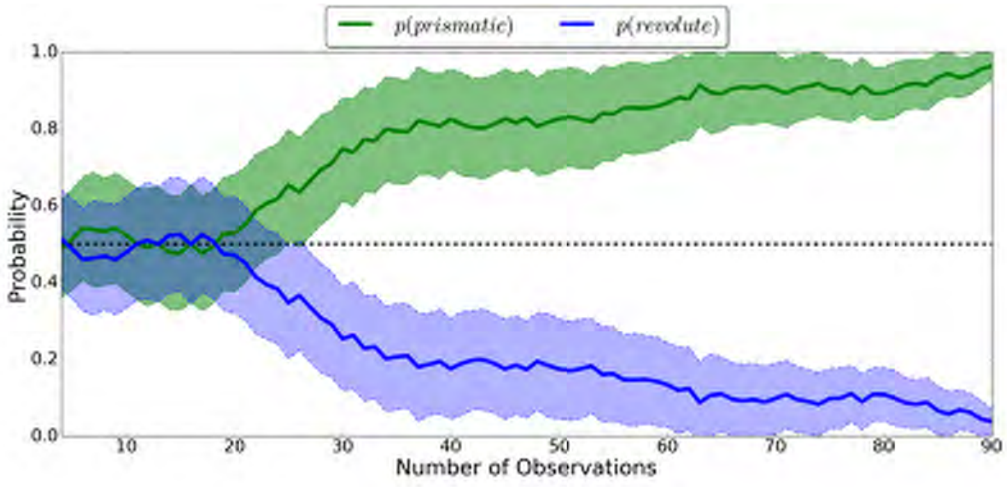


Figure 6.6: The upper graphic shows the evolution of the mean of the posterior probability of each model vs the number of observations when opening a door with a revolute model. The shadowed area, in light color, corresponds to a margin of two standard deviations. The middle image shows the evolution of the error. The bottom graphic shows the spatial evolution of the true model posterior during the opening motion of a revolute door



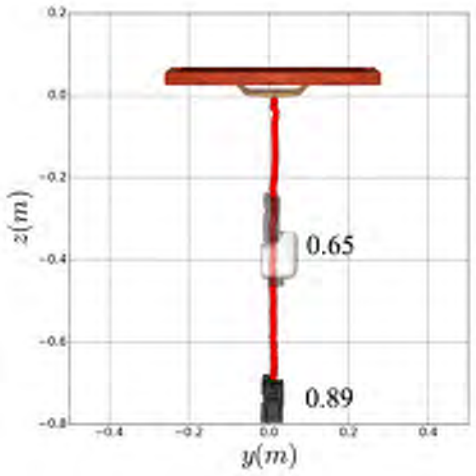


Figure 6.7: The upper graphic shows the evolution of the mean of the posterior probability of each model vs the number of observations when opening a door with a prismatic model. The shadowed area, in light color, corresponds to a margin of two standard deviations. The bottom graphic shows the spatial evolution of the true model posterior during the opening motion of a drawer

$$p\left(\mathcal{M}_{1+2} \mid \mathcal{D}_1, \mathcal{D}_2\right) > p\left(\mathcal{M}_1 \mid \mathcal{D}_1\right) \cdot p\left(\mathcal{M}_2 \mid \mathcal{D}_2\right)$$

This expression can be efficiently evaluated by using the BIC. Thus, it has to be checked:

$$BIC\left(\mathcal{M}_{1+2} \mid \mathcal{D}_1, \mathcal{D}_2\right) < BIC\left(\mathcal{M}_1 \mid \mathcal{D}_1\right) + BIC\left(\mathcal{M}_2 \mid \mathcal{D}_2\right)$$

Or equivalently:

$$-2 \log \mathcal{L} + k \log n < -2 \log(\mathcal{L}_1 \mathcal{L}_2) + k_1 \log n_1 + k_2 \log n_2 \tag{6.11}$$

with $\mathcal{L} = p\left(\mathcal{D}_1 \cup \mathcal{D}_2 \mid \mathcal{M}_{1+2}\right)$, $\mathcal{L}_1 = p\left(\mathcal{D}_1 \mid \mathcal{M}_1\right)$, $\mathcal{L}_2 = p\left(\mathcal{D}_2 \mid \mathcal{M}_2\right)$, and $n = n_1 + n_2$.

Intuitively, merging two models into one is beneficial if the joint model can explain the data equally well (i.e., $\mathcal{L} \approx \mathcal{L}_1 \mathcal{L}_2$), while requiring only a single set of parameters. It is checked whether merging the new trajectory with one of the existing models leads to a higher posterior compared to adding a new model for that trajectory to the set of previously encountered models. If more than two trajectories are considered, they have to be evaluated for the observation with all the recorded trajectories. This can become a process with very high computational cost. For this reason, in order to improve the performance, the recorded trajectories are classified according to the door classes provided by the presented perception approach. In this way, when opening a refrigerator door, the observations are only going to be compared with previous refrigerator door openings. Thus, the observed data is more likely to match the recorded data, and trajectories that are not likely to match, for instance a drawer, are not considered.

The procedure is summarized in Algorithm 6.2:

Algorithm 6.2 Model Selection Using Prior Knowledge

```
Input: New observed trajectory $\mathcal{D}_{new} = \left\{\mathbf{d}_j^{new}\right\}_1^N$ ;
  door class $c \in \{door,\ cabinet\ door,\ refrigerator\ door\}$;
  previously observed trajectories $\mathbb{D}_c = \{\mathcal{D}_s\}_1^S$
Output: Best model $\mathbb{M}_{best}$and prior-knowledge updated $\mathbb{D}_c$
  $\mathcal{M}_{new} \leftarrow Kinematic\_Model\left(\mathcal{D}_{new}\right)$
  $\mathbb{M}_{best} \leftarrow \{\mathcal{M}_{new}\}$ , $\mathbb{D}_c \leftarrow \mathbb{D}_c \cup \{\mathcal{D}_{new}\}$, $p_{best} \leftarrow 0$
  for all $\mathcal{D}_s \in \mathbb{D}$ do
      $\mathcal{M}_s \leftarrow Kinematic\_Model\left(\mathcal{D}_s\right)$
      $\mathcal{M}_{new+s} \leftarrow Kinematic\_Model\left(\mathcal{D}_{new} \cup \mathcal{D}_s\right)$
      if $p\left(\mathcal{M}_{new+s} \mid \mathcal{D}_{new}, \mathcal{D}_s\right) > p\left(\mathcal{M}_{new} \mid \mathcal{D}_{new}\right) p\left(\mathcal{M}_s \mid \mathcal{D}_s\right)$ & $p\left(\mathcal{M}_{new+s} \mid \mathcal{D}_{new}, \mathcal{D}_s\right) >$
  $p_{best}$ then
          $\mathbb{M}_{best} \leftarrow \{\mathcal{M}_{new},\ \mathcal{M}_s\}$
          $\mathbb{D}_c \leftarrow \{\mathcal{D}_1, \ldots, \mathcal{D}_{new} \cup \mathcal{D}_s \ldots, \mathcal{D}_S\}$
          $p_{best} \leftarrow p\left(\mathcal{M}_{new+s} \mid \mathcal{D}_{new}, \mathcal{D}_s\right)$
      end if
  end forreturn $\mathbb{M}_{best}$ and $\mathbb{D}_c$
```

After a set of models is identified as prior information, this knowledge can be exploited to improve the inference. If the newly observed data is merged with an existing model, the parameter vector is estimated from a much larger dataset $\mathcal{D}_j \cup \mathcal{D}_{new}$, instead of $\mathcal{D}_{new}$, which leads to a better estimation. However, since this process is carried out repeatedly as new observations are acquired, if the currently manipulated door ceases to be explained by the known models, the method instantaneously creates a new model. After door opening, the algorithm is repeated and the new data serves as additional prior information for the future merged with a previous trajectory or as a new trajectory.

6.7.1 Bias-Variance Trade-Off

Prior knowledge is expected to help guide the learning process when the robot is opening a door. However, it might also be a double-edged sword if the guidance is too rigid. Supposing the robot has previously opened a cabinet door with a revolute model, it should not think the next cabinet door it will operate will necessarily have a revolute model since it might also be prismatic. The guidance should neither be too soft since it is expected to boost its performance if the next cabinet door is revolute. In statistics it is known as the bias-variance trade-off (reference). In statistics and machine learning, the bias–variance tradeoff is the property of a set of predictive models whereby models with a lower bias have a higher variance and vice versa. Ideally, one wants to choose a model that both accurately captures the regularities in its training data, but also generalizes well to unseen data. Unfortunately, it is typically impossible to do both simultaneously. High-variance learning methods may be able to represent their training set well but are sensitive to noise. In contrast, algorithms with low variance typically produce simpler models that may underfit their training data, failing to capture important regularities [27].

Let's suppose a training set of points $\mathcal{X}$, and the true set of values associated to each training sample $\mathcal{Y}$. Let $\hat{f}(\mathcal{X})$ be the function that best approximates the true function $f(\mathcal{X})$ by means of a learned probabilistic model. A measure of the optimatility of the first function can be obtained by the mean square error method. This is equivalent to:

$$E\left[\left(\mathcal{Y}-\hat{f}(\mathcal{X})\right)^2\right] = \left[\left(\text{Bia}s\left[\hat{f}(\mathcal{X})\right]\right)^2 + \text{Variance}\left[\hat{f}(\mathcal{X})\right] + \sigma^2\right]$$

where σ^2 is the irreducible error due to uncertainty in measurement, and:

$$\text{Bia}s\left[\hat{f}(\mathcal{X})\right] = E\left[\hat{f}(\mathcal{X})\right] - f(\mathcal{X}) \quad ; \quad \text{Variance}\left[\hat{f}(\mathcal{X})\right] = E\left[\hat{f}(\mathcal{X})^2\right] - E\left[\hat{f}(\mathcal{X})\right]^2$$

Intuitively, the variance measures how much does $\hat{f}(\mathcal{X})$ move around its mean, while the square of the bias can be thought as the distance between the predictions and the true values. Therefore, a zero-bias model will perfectly fit the data but this usually involves a lot of movement around the mean, while a zero-variance model might not always exactly fit the points. That is the reason why there is a trade-off. The optimal balance is the one that minimizes the overall error. A graphical description of this trade-off is shown in Figure 6.8.

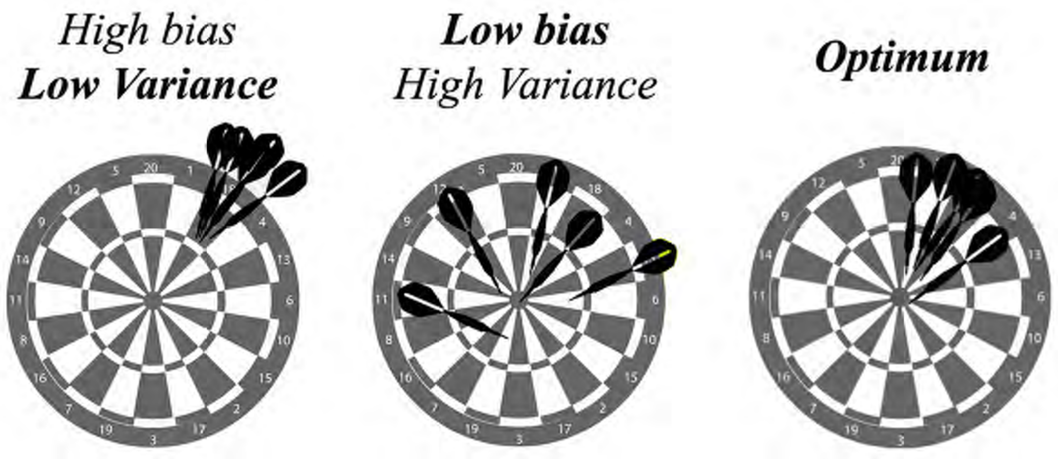


Figure 6.8: Bias-Variance Trade-Off

The bias-variance dilemma has been examined in the context of human cognition, most notably by Gigerenzer et al. [25] in the context of learned heuristics. They have argued that the human brain resolves the dilemma in the case of the typically sparse, poorly-characterised training-sets provided by experience by adopting high-bias/low variance heuristics. This reflects the fact that a zero-bias approach has poor generalisability to new situations, and also unreasonably presumes precise knowledge of the true state of the world. The resulting heuristics are relatively simple, but produce better inferences in a wider variety of situations.

Ideally the proposed probabilistic model for exploiting prior knowledge should both capture the recorded trajectories, but also be able to generalize well when a less-likely kinematic model appears. In the Bayesian formulation that is to have an adequate balance between the prior and the new observations when computing the posterior.

6.7.2 Results

For testing the robustness and performance of the learning framework, a series of experiments were conducted. The robot had to open two different doors with different kinematic models: a drawer and a revolute refrigerator door. Ten trials were done for each door in three different situations: when the prior knowledge is predominantly revolute, when it is predominantly prismatic and when both are balanced. For both doors, the convergence behavior of the model posterior, and for the revolute model door also the radius error was evaluated.

Results are shown in Figures 6.9, 6.10, 6.11, 6.12, 6.13, and 6.14.

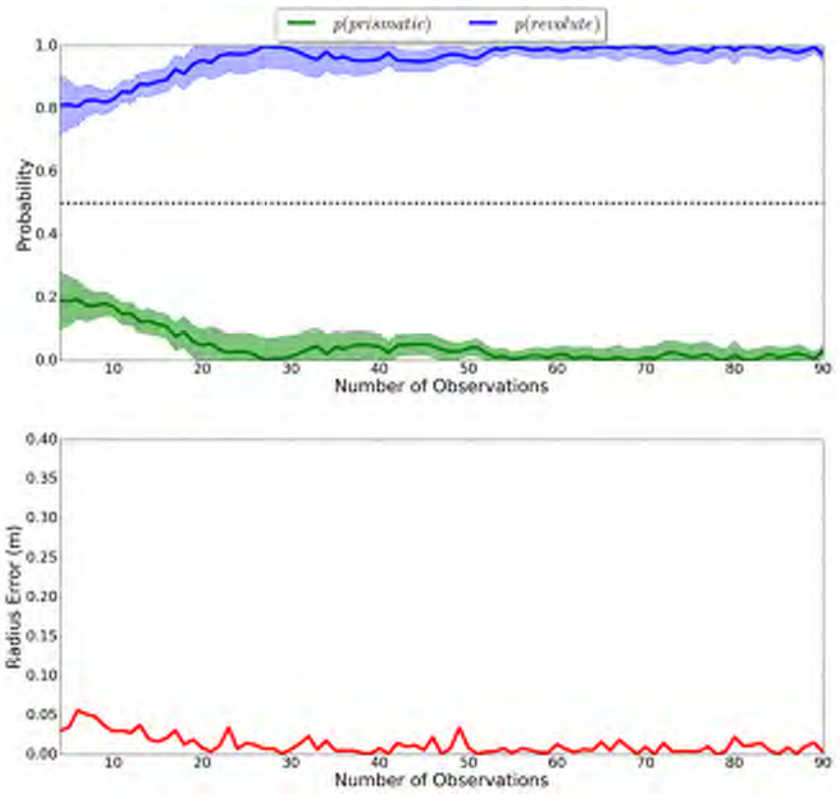


Figure 6.9: Evolution of the posterior and the error in the radius estimation when opening a revolute door with a prior predominantly revolute. In solid lines it is shown the mean for all the realizations. The shadowed area represents a margin of two standard deviations

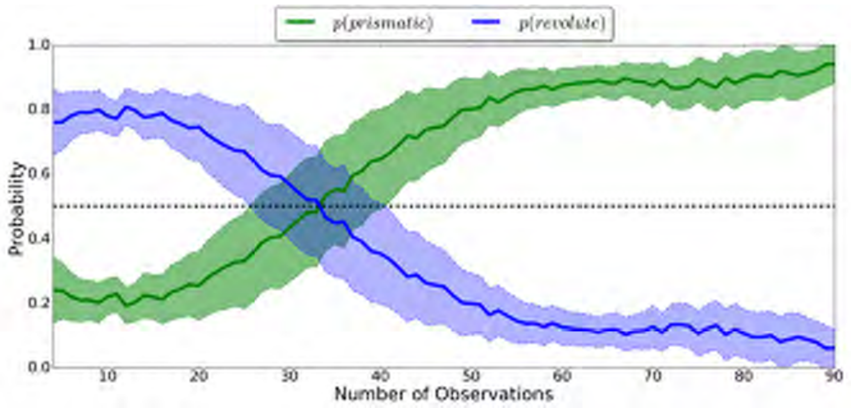
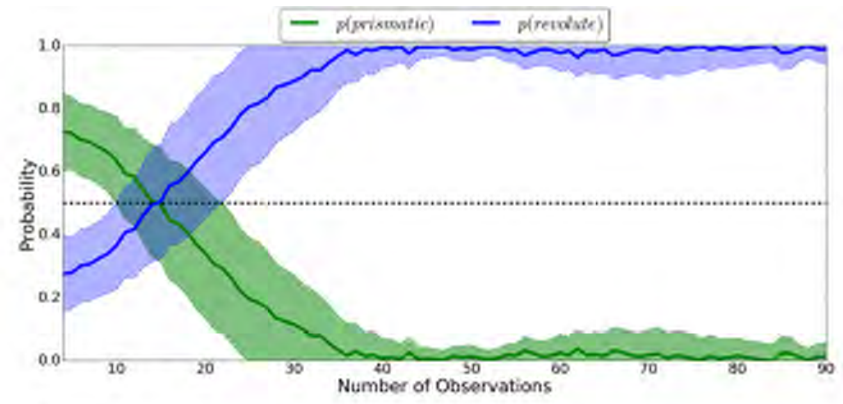


Figure 6.10: Evolution of the posterior when opening a prismatic model with a prior predominantly revolute. In solid lines it is shown the mean for all the realizations. The shadowed area represents a margin of two standard deviations



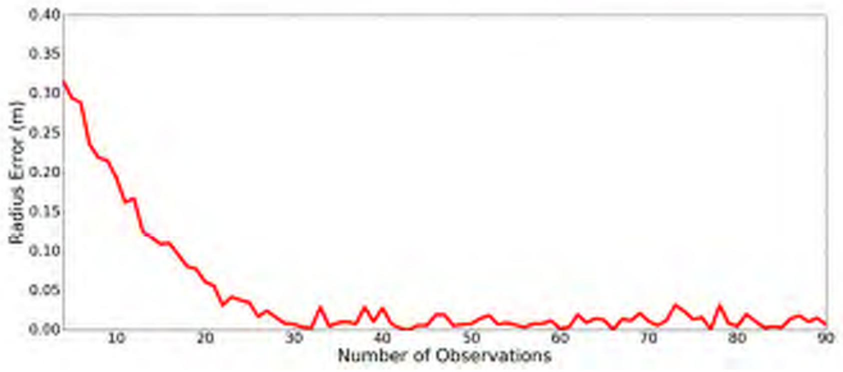


Figure 6.11: Evolution of the posterior and the error in the radius estimation when opening a revolute door with an equilibrated prior. In solid lines it is shown the mean for all the realizations. The shadowed area represents a margin of two satandard deviations

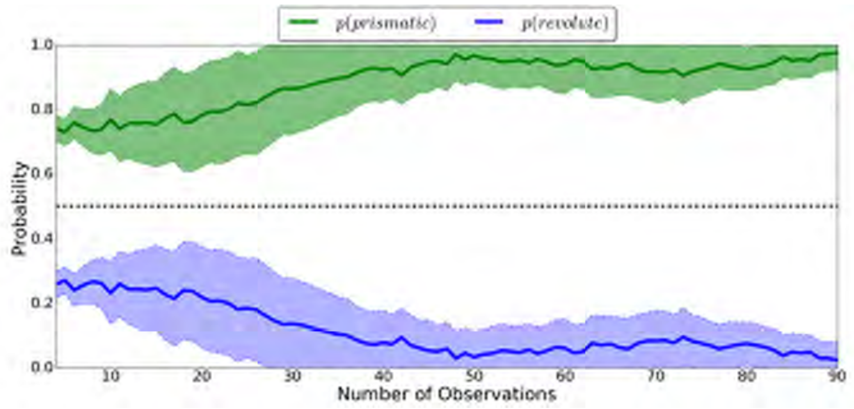


Figure 6.12: Evolution of the posterior when opening a prismatic door with an equilibrated prior. In solid lines it is shown the mean for all the realizations. The shadowed area represents a margin of two standard deviations

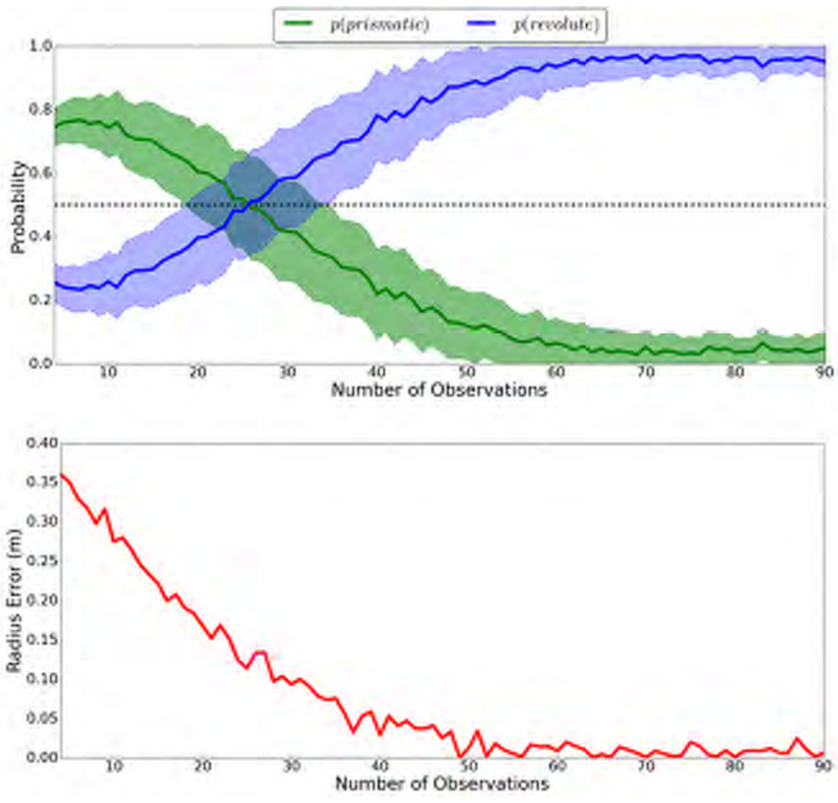


Figure 6.13: Evolution of the posterior and the error in the radius estimation when opening a revolute door with a prior predominantly prismatic. In solid lines it is shown the mean for all the realizations. The shadowed area represents a margin of two standard deviations

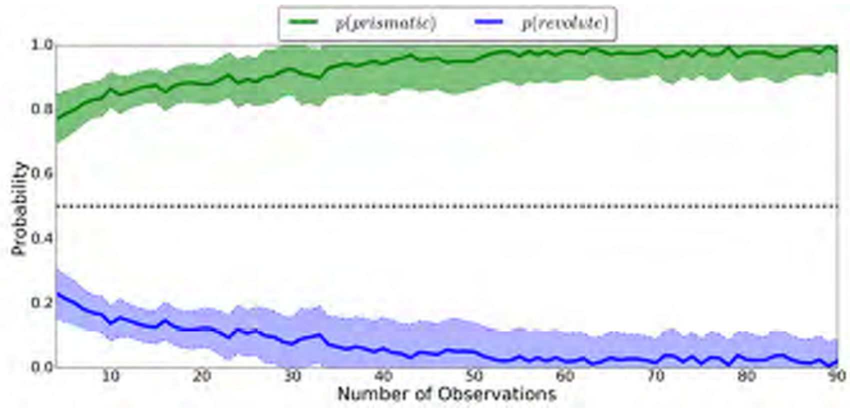


Figure 6.14: Evolution of the posterior when opening a prismatic door with a prior predominantly prismatic. In solid lines it is shown the mean for all the realizations. The shadowed area represents a margin of two standard deviations

Regarding the evaluation of the model posterior against the number of observations, it can be observed that the behavior depends on the predominant prior. When it matches the true model (Figure 6.9 and Figure 6.14), the posterior converges rapidly and remains at a high value for almost all the realizations. When it comes to the results obtained with a balanced prior (Figure 6.11 and Figure 6.12), the behavior depends on the true model. When few new observations are available they tend to merge with a previously recorded prismatic model. This is reasonable, since the trajectory is very similar for both models at this point but the complexity is penalized favoring the prismatic model. However at a relatively low number of observations, the posterior rapidly converges to the true model. Finally, in the case the prior does not match the true model, the behavior is symetric for both doors (Figure 6.10 and Figure 6.13). At the beginning, the observations merge with the previous recorded trajectories. However, when the number of observations is sufficiently large, it rapidly converges towards the true model.

Regarding the convergence behavior of the error in the radius estimation, it is clear that the prior knowledge improves it. As soon as the observations merge with a previous experience, the error decreases significantly.

It can be concluded from these results that the proposed learning approach allows the exploiting of prior knowledge to improve the model and the parameter estimation. Furthermore, it has been proved robust in the presence of biased priors. Also, it boosted the performance significatively when the predominant prior corresponded to the true model.

6.8 Learning from Human Demostrations

Humans have a great deal of intuition about how to accomplish many tasks that we would like robots to perform, but these intuitions do not necessarily translate naturally to code. Robot learning from demostration is a research paradigm that can play an important role in addressing the isue of scaling up robot learning [42]. Providing robots with the ability to learn the kinematic models of doors requires object detection, pose estimation and model learning. It is desirable for robots to learn these models from demonstrations provided by ordinary users. Therefore, the main requirement is a robust tracking of the door motion with the vision system. These demonstrations could then be exploited as prior knowledge. With the presented probabilistic framework, the only neccessary input is a set of observations of the handle position. In the previous section, this has been achieved exploiting the fact that the robot end-effector and the handle follow the same trajectory when the door is manipulated. However, using the proposed perception approach, this observations can be provided by the robot vision system.

To achieve a reliable handle tracking, the main issue that has to be adressed is, in case there are multiple handle detections, for which of the handles the kinematic model has to be estimated, i.e., which handle is moving. In order to solve the handle tracking problem, it is useful to represent the trajectory as a set of 3D points, i.e., as a point cloud. With the combination of a Voxel Grid Filter (section 4.2.1) and an Euclidean Clusterizer, a robust and efficient tracking can be achieved (Figure 6.15).

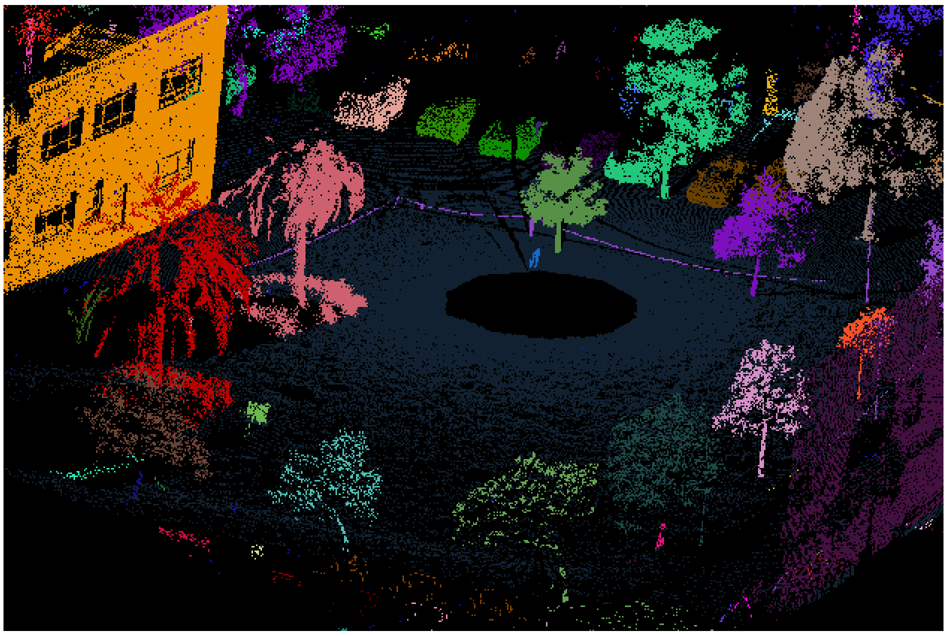


Figure 6.15: Point cloud clusterization using a Euclidean Clusterizer [21]

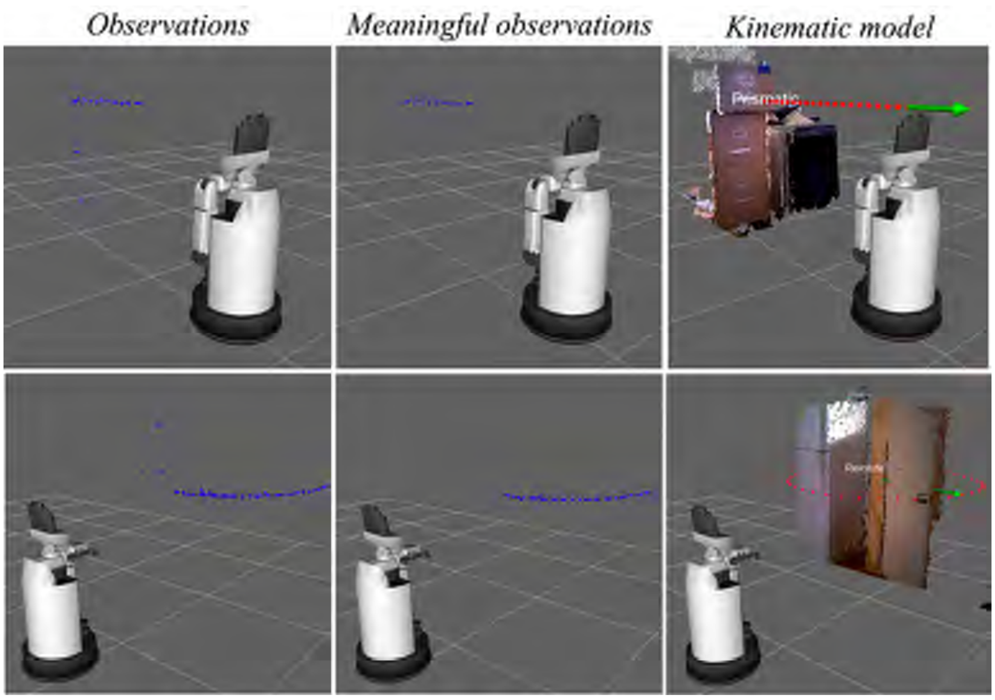


Figure 6.16: Learning kinematic models from human demonstrations

Clusterizing, in the context of a point cloud, means to group the points according to some criteria. Thus, an Euclidean Clusterizer groups points according to its euclidean distance. Formally, the clusters will be defined as follows:

> Let P be the point cloud of observations and $O_i = p_i \in P$ be a distinct point cluster from $O_j = p_j \in P$ if $\min \mid p_i - p_j \mid^2 \geq d_{th}$, where d_{th} is a maximum imposed Euclidean distance threshold.

The above equation states that if the minimum distance between a set of points $p_i \in P$ and another set $p_j \in P$ is larger than a given distance value, then the points in p_i are set to belong to a point cluster O_i and the ones in p_j to another distinct point cluster O_j. Then, since the handle trajectories are more than a certain distance threshold apart, each one is going to be in a different cluster.

Using a Voxel Grid filter the point density of the point cloud is constant over the space. Thus, the more a handle moves, its point cloud is going to be more spanned and then, more points will be associated to a moving handle cluster than a static handle cluster. Then, by getting the biggest cluster, the meaningful observations to estimate the kinematic model are acquired and an efficient handle tracking achieved.

The proposed approach was tested in a serie of experiments shown in Figure 6.16. The perception system was used successfully for the acquisition of the necessary data to infer the kinematic model of the door. The meaningful observations were extracted correctly from the set with all of them and the kinematic model was estimated accurately.

Thus, it is concluded that, with the proposed framework, the prior knowledge gathering process is simplifed as it can be provided from human demonstrations, improving in this way the learning capabilities of the robot.

6.9 Framework Extension: Building a Semantic Map

The evolution of contemporary mobile robotics has given thrust to a series of additional conjunct technologies, such as the semantic mapping, that provides an abstraction of the space [35]. Knowledge about the structure and the current state of the world is usually encoded in form of a map. Until know, these representations have focused mostly on the spatial structure of the environment. This kind of maps are needed for navigation but they do not contain the more qualitative type of information needed to perform task planning. This tendency is now changing, and the field of autonomous robotics is witnessing an increasing interest in the so-called semantic maps, which integrate semantic domain knowledge into traditional robot maps. These maps can provide a mobile robot with deduction abilities to infer information from its world [24].

In this project, when the robot does not know the kinematic model of the different types of doors it has to infere it from observations. But with the proposed learning framework, if the robot associates its prior knowledge to an spatial representation, i.e., build a semantic map, it could perform the door opening task more reliably and efficiently when operating in the same enviroment [57].

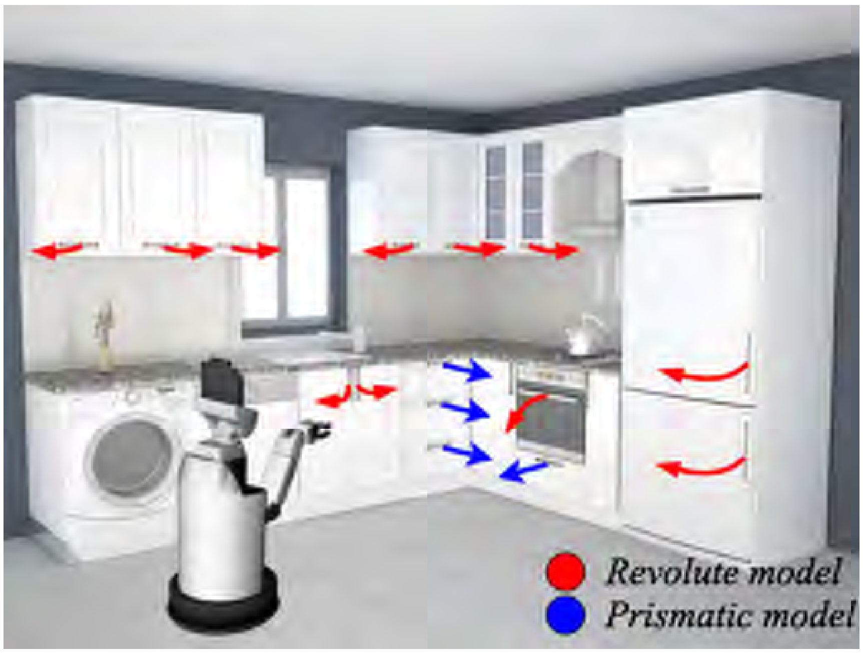


Figure 6.17: Semantic Map with different doors and their kinematic models

Then, the semantic map could be built while the robot operates different doors with the following procedure (Figure 6.17):

1. The user starts by specifying a coarse coordinate for a new cabinet to be opened and the robot retrieves the stored handle pose, navigates towards it and starts a detection to verify the stored handle pose. In case multiple handles are found, the one closest to the stored handle pose is selected.
2. Then, the robot grasps the handle and opens the door.
3. Finally, the generated trajectory is stored as semantic map spatial knowledge.

With the proposed framework extension, using the knowledge the robot has acquired operating the doors of a certain enviroment, any robot could be able to generate an open-loop trajectory and successfully open the door or drawer when required to execute the same operation.

7 Conclusions

This work started with a simple objective: enable service robots to open a general type of door. I couldn't imagine how many different fields of robotics research the exploration of that apparent simple question would involve. Every little advance came up after looking at the problem with a new perspective, proving that robotics lies in the intersection of computing, engineering, science, and mathematics.

Identifying the handle lied in the field of computer vision, and led to the exploration of hot topics such as deep learning. As a result, a state-of-the-art real-time CNN-based object detection method was built. It was able to robustly detect three door classes and their corresponding handles under variable conditions. However, manipulation not only needs the recognition capabilities, it requires the computation of relevant three-dimensional geometric features. Exploiting the nature of the data provided by the robot perception system, the CNN detections were combined with a novel point cloud processing algorithm that allowed the computation of the end-effector grasping pose in real-time.

The door manipulation problem led to a wide review of standards as well as state-of-the-art robot control and motion planning algorithms. From inverse kinematics, to impedance control, and finally the TSR framework. Combining these methods, an efficient door manipulation strategy was achieved.

Finally, the problem of dealing with the "a priori" uncertainty of the door kinematic model led to a probabilistic framework to learn them from the robot observations. Exploiting the Bayesian perspective, current robotics problems such as learning from experience and from human demonstrations were addressed. Also, a semantic map that enables a more efficient task reasoning can be built. As a result, by means of the Toyota HSR platform, the aim of the project was finally achieved: a unified framework to robustly operate all possible kinds of doors with a service robot.

Some of the most cutting-edge technologies available today are coming together in a way that gives service robots more capabilities than ever before. With expanding capabilities, service robots will become an increasingly common presence in working and home spaces around the world.

The exploration of a simple question reveals the magic of robotics research: there is always room for improvement and the limits of what is possible are always being long lifted. It is still a developing field that it is undoubtley advancing rapidly. Allowing room for a combination of extremely creative thinking and seeing your creation spring to life, makes robotics, undoubtely, a fascinating field.

Bibliography

[1] Aho, K., Derrberry, D., and Peterson, T. Model selection for ecologists: the worldviews of AIC and BIC. *Ecology 95*, 3 (2014), 631–636.

[2] Alenya, G., Foix, S., and Torras, C. Using ToF and RGBD cameras for 3D robot perception and manipulation in human environments. *Intelligent Service Robotics 7*, 4 (2014), 211–220.

[3] Anguelov, D., Koller, D., Parker, E., and Thrun, S. Detecting and modeling doors with mobile robots. *Proc. of IEEE International Conference on Robotics and Automation (ICRA), New Orleans, LA, USA* (2004).

[4] Arduengo, M. Labelled image dataset for door and handle detection. https://github.com/MiguelARD/DoorDetect-Dataset.

[5] Asada, H., and Leonard, J. Introduction to robotics. https://ocw.mit.edu/courses/mechanical-engineering/2-12-introduction-to-robotics-fall-2005/lecture-notes/, MIT OpenCourseWare, 2005.

[6] Banerjee, N., Long, X., Du, R., Polio, F., Feng, S., Atkeson, C., Gennert, M., and Padir, T. Human-supervised control of the ATLAS humanoid robot for traversing doors. *Proc. of IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)* (2015), 722–729.

[7] Berenson, D. Constrained manipulation planning. Tech. rep., PhD Dissertation, The Robotics Institue, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2011.

[8] Berenson, D., Srinivasa, S., and Kuffner, J. Task Space Regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research 30*, 12 (2011), 1435–1450.

[9] Borgsen, S., Schoepfer, M., Ziegler, L., and Wachsmuth, S. Automated door detection with a 3D-sensor. *Proc. Canadian Conference on Computer and Robot Vision (CRV), Montreal, Quebec, Canada* (2014).

[10] Burnham, K., and Anderson, D. Multimodel inference: Understanding AIC and BIC in model selection. *Sociological Methods Research* (2004).

[11] Buss, S., and Kim, J.-S. Selectively damped least squares for inverse kinematics. *Journal of Graphics Tools 10*, 3 (2005), 37–49.

[12] Chen, W., Qu, T., Zhou, Y., Weng, K., Wang, G., and Fu, G. Door recognition and deep learning algorithm for visual based robot navigation. *Proc. of IEEE International Conference on Robotics and Biomimetics (ROBIO)* (2014), 1793–1798.

[13] Chitta, S., Cohen, B., and Likhachev, M. Planning for autonomous door opening with a mobile manipulator. *IEEE Interntional Conference on Robotics and Automation* (2010), 1799–1806.

[14] Choi, S., Kim, T., and Yu, W. Performance evaluation of RANSAC family. *BMVC* (2009).

[15] Christensen, H., and Hager, G. Sensing and estimation, chapter 5. In *Springer Handbook of Robotics, 2nd ed.* (2016), B. Siciliano and O. Khatib, Eds., Springer Verlag.

[16] Corke, P. *Robotics, vision and control, 2nd ed.* Springer International, 2011.

[17] D. Nguyen-Tuong. Model learning in robot control. Tech. rep., PhD Dissertation, Technischen Fakultat der Albert-Ludwigs-Universitat Freiburg im Breisgau, 2011.

[18] Deng, L. Three classes of deep learning architectures and their applications: A tutorial survey. *APSIPA Transactions on Signal and Information Processing* (2012).

[19] Deng, L. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing 3* (2014).

[20] Diankov, R., Srinivasa, S., Ferguson, D., and Kuffner, J. Manipulation planning with caging grasps. *Proc. of IEEE-RAS International Conference on Humanoid Robots* (2008).

[21] Douillard, B., Underwood, J., Kuntz, N., Vlaskine, V., Quadros, A., Morton, P., and Frenkel, A. On the segmentation of 3D LIDAR point clouds. *IEEE International Conference on Robotics and Automation (ICRA)* (2011), 2798–2805.

[22] Enders, F., Trinkle, J., and Burgard, W. Learning the dynamics of doors for robotic manipulation. *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2013), 3543–3549.

[23] Ferreira, J., and Dias, J. *Probabilistic approaches for robotic perception.* Springer International Publishing, 2014.

[24] Galindo, C., Fernandez-Madrigal, J., Gonzalez, J., and Saffiotti, A. Robot task planning using Semantic Maps. *Robotics and Autonomous Systems 56* (2008), 955–966.

[25] Gigerenzer, G., and Gaissmaier, W. Heuristic Decision Making. *Annual Review of Psychology 1*, 62 (2011), 451–482.

[26] HASHIMOTO, K., SAITO, F., YAMAMOTO, T., AND IKEDA, K. A field study of the Human Support Robot in the home environment. *Proc. of 2013 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO), Tokyo, Japan* (2013).

[27] HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. *The elements of statistical learning, 2nd ed.* Springer Verlag, 2009.

[28] HU, F., ZHAO, Y., WANG, W., AND HUANG, X. Discrete point cloud filtering and searching based on VGSO algorithm. *Proc. of the European Council for Modeling and Simulation (ECMS)* (2013).

[29] JAIN, A., AND KEMP, C. Pulling open novel doors and drawers with equilibrium point control. *Proc. of 9th IEEE-RAS International Conference on Humanoid Robots* (2009).

[30] JAUREGI, E., MARTINEZ-OTZETA, J., SIERRA, B., AND LAZKANO, E. Door hanle identification: A three-stage approach. *Proc. of 6th IFAC Symposium on Intelligent Autonomous Vehicles 40*, 15 (2007), 517–522.

[31] KARAYIANNIDIS, Y., SMITH, C., VINA, F., OGREN, P., AND KRAGIC, D. Model-free robot manipulation of doors and drawers by means of fixed-grasps. *Proc. of IEEE International Conference on Robotics and Automation (ICRA)* (2013).

[32] KELLY, A. *Mobile Robotics.* Cambridge University Press, 2013.

[33] KEMP, C., EDSINGER, A., AND TORRES-JARA, E. Challenges for robot manipulation in human environments. *IEEE Robotics Automation Magazine 14*, 1 (2007).

[34] KESSENS, C., RICE, J., SMITH, D., BIGSS, S., AND GARCIA, R. Utilizing compliance to manipulate doors with unmodeled constraints. *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2010), 483–489.

[35] KOSTAVELIS, I., AND GASTERATOS, A. Semantic mapping for mobile robotics tasks: A survey. *Robotics and Autonomous Systems 66* (2011), 86–103.

[36] KUZNETSOVA, A., ROM, H., ALLDRIN, N., UIJLINGS, J., KRASIN, I., PONT-TUSET, J., KAMALI, S., POPOV, S., MALLOCI, M., DUERIG, T., AND FERRARI, V. The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. *ArXiv e-prints* (2018). arXiv:1811.00982v1.

[37] KWAK, N., ARISUMI, H., AND YOKOI, K. Visual recognition of a door and its knob for a humanoid robot. *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)* (2011), 2079–2084.

[38] LATOMBE, J. *Robot motion planning.* Springer Science+Business Media, LLC, 1991.

[39] LAVALLE, S. *Planning algorithms.* Cambridge University Press, 2006.

[40] LaValle, S. Motion planning (Part I): The essentials. *IEEE Robotics Automation Magazine 18*, 1 (2011), 79–89.

[41] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature 521*, 7553 (2015), 436–444.

[42] Lee, J. A survey of robot learning from demostrations for Human-Robot collaboration. *ArXiv e-prints* (2017). arXiv:1710.08789v1.

[43] Leon, B., Morales, A., and Sancho-Bru, F. *From robot to human grasping simulation.* Springer International, 2014.

[44] Llopart, A., Ravn, O., and Andersen, N. Door and cabinet recognition using convolutional neural nets and real-time method fusion for handle detection and grasping. *Proceedings of 2017 3rd International Conference on Control, Automation and Robotics, ICCAR* (2017).

[45] Lutscher, E., Lawitzky, M., Cheng, G., and Hirche, S. A control strategy for operating unknown constrained mechanisms. *Proc. of IEEE International Conference on Robotics and Automation (ICRA)* (2010), 819–824.

[46] MacKay, D. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003.

[47] Mason, M. Toward robotic manipulation. *Annual Review of Control, Robotics, and Autonomous Systems*, 1 (2018), 1–28.

[48] Meeussen, W., Wise, M., Glaser, S., Chitta, S., McGann, C., Mihelich, P., Marder-Eppstein, E., Muja, M., Erhimov, V., Foote, T., Husu, J., Rusu, R., Marthi, B., Bradski, G., Konolige, K., Gerkey, B., and Berger, E. Autonomous door opening and plugging in with a personal robot. *Proc. of the IEEE International Conference on Robotics and Automation* (2010).

[49] Morrison, D., Corke, P., and Leitner, J. Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. *ArXiv e-prints* (2018). arXiv:1804.05172v2.

[50] Nagatani, K., and Yuta, S. Designing strategy and implementation of mobile manipulator control system for opening door. *Proc. of IEEE International Conference on Robotics and Automation (ICRA)* (1996).

[51] Nemec, B., Zlajpah, L., and Ude, A. Door opening by joining reinforcement learning and intelligent control. *Proc. of the 18th International Conference on Advanced Robotics (ICAR)* (2017).

[52] Nguyen-Tuong, D., and Peters, J. Model learning for robot control: A survey. *Cognitive Process 4*, 12 (2011), 319–340.

[53] Niemeyer, G., and Slotine, J. A simple strategy for opening an unknown door. *Proc. of the IEEE International Conference on Robotics and Automation 2* (1997).

[54] Oehler, B., Stueckler, J., Welle, J., Schulz, D., and Behnke, S. Efficient multi-resolution plane segmentation of 3D point clouds. *Proc. of the 4th International Conference on Intelligent Robotics and Applications (ICIRA)* (2011).

[55] OpenImages. Overview of Open Images V4. https://storage.googleapis.com/openimages/web/factsfigures.html.

[56] Ott, C., Bauml, B., Borst, C., and Hirzinger, G. Employing cartesian impedance control for the opening of a door: A case study in mobile manipulation. *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Workshop on mobile manipulators* (2005).

[57] Pangercic, D., Pitzer, B., Tenorth, M., and Beetz, M. Semantic Object Maps for robotic housework - representation, acquisition and use. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2012), 4644–4651.

[58] Paul, R. *Robot Manipulators: Mathematics, Programming, and Control.* The MIT Press, 1981.

[59] Peterson, L., Austin, D., and Kragic, D. High-level control of a mobile manipulator for door opening. *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems* (2000).

[60] Quintana, B., Prieto, S., Adan, A., and Bosch, F. Door detection in 3D coloured point clouds of indoor environments. *Automation in Construction 85* (2018).

[61] Redmond, J., Divvala, S., Grirshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. *ArXiv e-prints* (2016). arXiv:1506.02640v5.

[62] Redmond, J., and Farhadi, A. YOLO9000: Better, faster, stronger. *ArXiv e-prints* (2016). arXiv:1612.08242v1.

[63] Redmond, J., and Farhadi, A. YOLOv3: An incremental improvement. *ArXiv e-prints* (2018). arXiv:1804.02767v1.

[64] Ruhr, T., Sturm, J., Pangercic, D., Beetz, M., and Cremers, D. A generalized framework for opening doors and drawers in kitchen environments. *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)* (2012).

[65] Russell, S., and Norvig, P. *Artificial Intelligence: A Modern Approach, 3nd ed.* Prentice Hall, 2010.

[66] Rusu, R. *Semantic 3D object maps for everyday robot manipulation.* Springer Verlag, 2013.

[67] Rusu, R., and Cousin, S. 3D is here: Point Cloud Library (PCL). *Proc. of IEEE International Conference on Robotics and Automation (ICRA), Shangai, China* (2011).

[68] Rusu, R., Meeussen, W., Chitta, S., and Beetz, M. Laser-based perception for door and handle identification. *Proc. of International Conference on Advanced Robotics (ICAR)* (2009).

[69] Saha, S. *Introduction to Robotics, 2nd ed.* McGraw Hill Education India, 2014.

[70] Schnable, R., Wahl, R., and Klein, R. Efficient RANSAC for point cloud shape detection. *Computer Graphics 26*, 2 (2007).

[71] Shalaby, M., Salem, M.-M., Khamis, A., and Melgani, F. Geometric model for vision-based door detection. *Proc. of the 9th International Conference on Computer Engineering Systems (ICCES), Malaysia* (2014).

[72] Song, P., Yu, Y., and Zhang, X. Impedance control of robots: An overview. *Proc. of 2nd International Conference on Cybernetics, Robotics and Control* (2017).

[73] Sprenger, M., and Mettler, T. Service robots. *Business and Information Systems Engineering 57* (2015).

[74] Sturm, J., , Jain, A., Stachniss, C., Kemp, C., and Burgard, W. Operating articulated objects based on experience. *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Taiwan* (2010).

[75] Sturm, J. *Approaches to probabilistic model learning for mobile manipulation robots.* Springer Verlag, 2013.

[76] Sturm, J., Stachniss, C., and Burgard, W. A probabilistic framework for learning kinematic models of articulated objects. *Journal of Artificial Intelligence Research 41* (2011), 477–526.

[77] Taylor, L., and Nitschke, G. Improving Deep Learning using generic Data Augmentation. *ArXiv e-prints* (2017). arXiv:1708.06020v1.

[78] Torr, P., and Zisseman, A. MLESAC: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding* (2000).

[79] Torras, C. Service Robots for Citizens of the Future. *European Review 24*, 1 (2016), 17–30.

[80] Wang, H., Yang, B., and Chen, W. Unknown constrained mechanisms operation based on dynamic interactive control. *CAAI Transactions on Intelligence Technology 1*, 3 (2016), 259–271.

[81] Wang, Z., Liu, H., Qian, Y., and Xu, T. Real-time plane segmentation and obstacle detection of 3D point cloud for indoor scenes. *Proc. of ECCV* (2012).

[82] Welschehold, T., Dornhege, C., and Burgard, W. Learning mobile manipulation actions from human demonstrations. *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017).

[83] Wieland, S., Gonzalez-Aguirre, D., Vahrenkamp, N., Asfour, T., and Dillmann, R. Combining force and visual feedback for physical interaction task in humanoid robots. *Proc. of 9th IEEE-RAS International Conference on Humanoid Robots, Paris, France* (2009).

[84] Yamamoto, T., Nishino, T., Kajima, H., and Ohta, M. Human Support Robot (HSR). *Proceedings of SIGGRAPH'18 Emerging Technologies, Vancouver, BC, Canada* (2018).

[85] Yuan, T., Hashim, F., Zaki, W., and Huddin, A. An automated 3D scanning algorithm using depth cameras for door detection. *Proc. Electronics Symposium: Emerging Technology in Electronic and Information* (2015).

[86] Zhao, Z.-Q., Xu, S.-T., and Wu, X.-D. Object detection with deep learning: A review. *ArXiv e-prints* (2018). arXiv:1807.05511v1.

[87] Zuliani, M. RANSAC for dummies. Tech. rep., UCSB Vision Research Lab, 2017.