Detecting, Opening and Navigating through Doors: A Unified Framework for Human Service Robots

Francesco Savarese[1,2], Antonio Tejero-de-Pablos[2], Stefano Quer[1] and Tatsuya Harada[2]
[1]*DAUIN Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy*
[2]*Machine Intelligence Lab., The University of Tokyo, Tokyo, Japan*

Keywords: Service Robotics, Door Opening, State Machines, Object Detection, Autonomous System.

Abstract: For an autonomous robotic system, detecting, opening, and navigating through doors remains a very challenging problem. It involves several hard-to-solve sub-tasks such as recognizing the door, grasping the handle, discriminating between pulling or pushing the door, and detecting locked doors. Previous works tackle individual sub-problems, assuming that the robot is already facing the door handle or that the robot knows in advance the exact location of the door. However, ignoring the navigation through the door, using specialized robots, or specific types of doors, reduce the applicability of existing approaches. In this paper, we present a unified framework for the door opening problem, by taking a navigation scenario as a reference. We implement specific algorithms to solve each sub-task, and describe the hierarchical automata which integrates the control of the robot during the entire process. Moreover, we implement error recovery mechanisms to add robustness and to guarantee a high success rate. We carry out experiments on a realistic scenario using a standard service robot, the Toyota Human Support Robot. We show that our framework can successfully detect, open, and navigate through doors in a reliable way, with low error rates, and without adapting the environment to the robot. Our experiments demonstrate the high applicability of our framework.

1 INTRODUCTION

First attempts of human-robot cooperation focused on robots capable of guiding people in public environments like museums (Burgard et al., 1998; Kim et al., 2004; Thrun et al., 1999). However, influenced by the aging population problem, current service robotics is mainly focusing on the design of robots to assist elderly people, or people with mobility impairments, in their daily life at home (Khatib, 1999). Nowadays, robots are able to work in environments like houses or offices to perform common tasks such as picking up objects or delivering articles. They have also reached a high level of human-robot cooperation (Hernandez et al., 2017; Johnson et al., 2015; Krotkov et al., 2017; Lim et al., 2018). Overall, current approaches emphasize the ability to autonomously navigate unknown environments and to interact with humans.

To freely navigate in unmodified domestic environments, robots have to be able to perform basic obstacle avoidance and handle complex situations. In particular, one very common and still unsolved problem is opening a door, without the human assistance. Door opening has drawn attention because of its complexity, and because it involves different sub-tasks such as handle recognition, handle grasping, discrimination between pulling or pushing the door, and the detection of locked doors.

1.1 Related Works

Recent works have approached the problem of door opening using robots designed or specifically modified for the target. For example (Rhee et al., 2004) adopts an exclusive robot whose hand is specifically designed for the door opening task. Other approaches, e.g., (Aude et al., 2006; Chitta et al., 2010; Ott et al., 2007), detect doors and handles relying on data fusion information coming from cameras, lasers, and other sensors. Andreopoulos et al. (Andreopoulos and Tsotsos, 2007) tried to solve the door opening problem using a robotics wheelchair. They used a computer vision approach based on Viola-Jones for door and handle recognition. However, they only studied handle detection and grasping, without proposing a method for door opening. Boston Dynamics[1]

[1]https://www.bostondynamics.com/spot-mini.

Savarese, F., Tejero-de-Pablos, A., Quer, S. and Harada, T.
Detecting, Opening and Navigating through Doors: A Unified Framework for Human Service Robots.
DOI: 10.5220/0007947604160427
In *Proceedings of the 14th International Conference on Software Technologies (ICSOFT 2019)*, pages 416-427
ISBN: 978-989-758-379-7

presented a solution based on the cooperation of two SpotMini robots. However, given the robot structure (i.e., a four-legged robot), it is hard to transfer the approach to common service robots. Moreover, their approach is not public.

Many works concentrate on independent tasks of the door opening problem, often neglecting navigation issues. For example, the research in (Petrovskaya and Ng, 2007; Rhee et al., 2004; Peterson et al., 2000; Dongwon et al., 2004) only tackles handle unlatching and door opening, obviating approaching the door and navigating through it. For that reason, their major limitation lies on the premise that the robot is initially facing the door to detect the handle. The position of the robot with respect to the door can influence significantly the success of the detection process, meaning that the robot needs to know the door position in the space to proceed correctly. Thus, they are not suitable for realistic scenarios in which the robot is moving.

The challenging task of door opening while navigating has also received a lot of attention. Meeussen et al. (Meeussen et al., 2010) propose a framework that integrates autonomous navigation and door opening. For door detection, they use a point cloud representation, while for handle recognition, they combine laser scans and a computer vision approaches. Although they analyzed the entire navigation and door opening problem, their approach requires the knowledge of several details on the environment, such as the door width and the door type. Similar considerations can be made for Chitta et al. (Chitta et al., 2010), where a planning algorithm is proposed for opening (pulling and pushing) doors, but the robot needs to know in advance if the target door is a pulling or a pushing one. As the first task to solve to open a door is to find the door and detect the handle, Kim et al. (Kim et al., 2011) solve the detection task using a video cameras. However the proposed method detects doors using a context-based object recognition approach, limiting its applicability to well known environments. Shalaby et al. (Shalaby et al., 2014) base their recognition task completely on a vision system. The task is accomplished pairing visual information and door geometric description. However, the approach requires a prior knowledge of doors details (such as the handle height) limiting the method applicability to only well-known scenarios. Rusu et al. (Rusu et al., 2009) use a laser perception-based to robustly estimate the handle position. Klingbeil et al. (Klingbeil et al., 2010) combine a visual algorithm with laser data to locate the handle in the space. However, after handle unlatching, they do not tackle the problem of door opening. Jain et al. (Jain and Kemp, 2008) roughly estimate the handle position using a laser scan. After that, the robot haptically searches for the door handle over the surface of the door. After the handle unlatching, the door is pushed to be opened. They do not study the case of pulling door and they do not move the robot through the door. Gray et al. (Gray et al., 2013) focus their attention to both non-spring and spring-loaded doors. They propose a graph-based planning algorithm for opening both pulling and pushing doors. However, they do not analyze the entire problem flow. Moreover, their opening strategy requires to store additional information about the doors. Fernández et al. (Fernández-Caramés et al., 2014) proposed a method for detecting doors in corridors by detecting vertical lines from images obtained with a laser scan. The goal of this method is not opening the door (since it cannot detect handles), but to use the detected doors as a reference to correct the robot position in the map.

1.2 Contributions

The tasks required in a robust unified pipeline for door opening and navigation that does not rely in prior knowledge of the door characteristics are: Door and handle detection, door type/opening direction estimation, handle grasping, and finally navigation through the door. None of the aforementioned works tackle this pipeline in its entirety. Moreover, while an off-the-shelf system is desirable, most of them use custom-made robots that hinder reproducibility.

In this work, we present a unified framework to open doors while navigating the environment. We suppose no prior knowledge of the characteristics of the doors. We also recognize whether the door has to be pushed or pulled, and we perform appropriate actions to open it. We present a detailed hierarchical automata model of our framework. Using this model, we decompose the overall task into sub-tasks, and we perform proper error recovery during all main phases. We solve the implied sub-problems adopting a unified approach, providing detailed explanations of the resulting automata. For door and handle location estimation, we leverage a deep learning approach for automatic detection of doors and handles. In order to train such a detector, we contribute with the "MIL-door" data-set[2]. This approach allows our robot to recognize doors and handles even while navigating unknown environments, that is, without previously knowing their existence. After the door and the handle are detected, depth images are used to evaluate the location of the handle more precisely.

We assume the robot navigates an unmodified house, that is, a house furnished with common fur-

[2]https://www.mi.t.u-tokyo.ac.jp/projects/mildoor.

niture pieces and with non-automatic doors. In our framework all door characteristics, i.e., door width, handle position, and opening direction, are estimated at run-time. Moreover, our approach performs automatic door type detection (pushing or pulling) which is particularly important in real world scenarios. Our proposed framework also considers robot navigation in a structured environment, admitting semantic navigation. This allows studying the door opening problem from the perspective of a realistic navigation problem. To evaluate our framework, we chose a complex task among the *Robocup 2018*[3] challenges (i.e., the “Help Me Carry” task).

The proposed solution is more appropriate for real applications than the all aforementioned previous works. Moreover, we implemented it into a standard general purpose robot, namely, the Toyota Human Support Robot (HSR)[4], whereas the majority of the proposed solutions use specific architectures, such as the Personal Robot 2 (PR2) robotic platform, developed by Willow Garage[5]. We present extensive experimentation using a standard domestic robot platform in a realistic scenario, and we show the high applicability of our approach.

2 CONFIGURATION

2.1 Hardware Platform

As our development platform, we used the Toyota *Human Support Robot* (HSR). The robot is aimed at helping elderly people and people with disabilities. Given its design, HSR is optimal for operating in home settings without any modification that facilitates its tasks (e.g., automatic doors). Toyota also provides some primitives and some basic software routine for controlling the robot.

The HSR body is cylindrical with a set of wheels that makes the robot movable in all directions. It is equipped with a folding arm capable of grabbing objects, manipulating handles and even grasping paper sheets from the floor. Thanks to its microphone array and its speakers, HSR is able to receive voice commands and communicate with the user. Several sensors allow the robot interacting with the surrounding environment. The HSR head is equipped with a stereo video camera and a depth camera. The robot base is equipped with a collision detector. The Robot Operating System (ROS)[6] is installed on the robot, allowing communicating with the hardware layer. This way, writing low level controlling algorithms is not necessary.

[3]http://www.robocup2018.com.

[4]https://www.toyota-global.com/innovation/partner_-robot/robot/#link02.

[5]http://www.willowgarage.com.

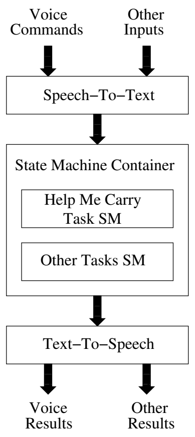


Figure 1: Our Robot Software Architecture consists of three layers: A speech to text layer for command processing, a state machine container layer that activates state machines according to the task, and a text to speech layer for result conveying.

2.2 Software Architecture

Figure 1 shows our software architecture. We designed it to implement the robot’s functionality, and it is the backbone of the entire system. It allows managing several basic tasks, the human-robot interaction, and easily adding new functionality on-demand (e.g., replacing voice commands with visual QR-code inputs). This improves system versatility, but it is not essential for the paper’s goal.

We defined three different layers:

- A command processing layer (speech-to-text). We use the HSR’s microphone array to capture the user command, and then we internally process it.
- A container (state machine container). State machines are deployed to solve different tasks.
- A user-friendly communication layer (text-to-speech). This is used to convey the operation results to the user.

The first layer processes the user’s voice command, and it forwards the result to the second layer. To interpret the voice command, and generate a command, we used the Google Cloud Speech-to-Text API[7]. This tool allows developers to convert speech into text exploiting the power of neural networks and using the Google Cloud suite. Depending on the given command, the second layer activates the proper state ma-

[6]http://www.ros.org.

[7]https://cloud.google.com/speech-to-text.

chine to execute the task required by the user. The third layer receives the results of the state machines, which are interpreted and communicated to the user in a user-friendly fashion. The state machine container is the element that provides flexibility to the entire architecture. It is possible, in fact, to embed new state machines for executing tasks. We implement all state machines using SMACH[8].

2.3 Semantic Navigation Framework

For the path planning we rely on the ROS global and local path planners. These modules receive the desired coordinates in the space, and they convert these coordinates into commands to move the robot. Using the ROS navigation stack built-in Hector-SLAM algorithm (Kohlbrecher et al., 2011) we can create a map describing the environment and the obstacles. This map allows the robot to receive coordinates and reach specific locations by automatically choosing an optimal path free of obstacles. However, semantic navigation requires a richer description of the environment to convert human understandable locations (e.g., *the kitchen table*) into suitable coordinates for the robot. As a consequence, additional information needs to be added to the map to improve the knowledge about the environment. We propose a framework for creating and managing semantic maps. This framework works as an interface layer, converting the location sent by the user to a location understandable by the motion planning module. Using RVIZ[9] we manually associate coordinates in the path planner map to human understandable locations. The association among coordinates and locations are stored as metadata into an *xml*, and a *csv* files.

We manage two different types of entities in the environment: *Rooms* and *locations*. A room is a portion of the map identified by walls or boundaries. Locations are places inside rooms. Each room can contain multiple locations. A room entity is identified by its name and it is represented by a list of corners, arranged as a polygon, plus a room center. To manage polygons and coordinates we use the python package *matplotlib.path*. A location, on the other hand, is represented by a location name, its coordinates in the map and some attributes describing the place (e.g., "isStorage" is a Boolean attribute stating if the location is a storage area). The hierarchical relationship between rooms and locations are stored in *xml* format while the room and location names with their respective coordinates are stored in *csv* format.

[8]small SMACH is a ROS-independent Python library for building hierarchical state machines.

[9]RVIZ is a tool for displaying sensor data using ROS.

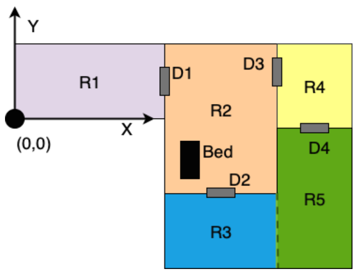


Figure 2: Example of a map for the navigation environment, with rooms (R), doors (D), and locations (Bed).

Figure 2 is a graphical representation of a possible environment map, where R1–R5 designate rooms and D1–D4 indicate doors. R3 and R5 are not separated by a wall. The position of elements in the map is retrieved with respect a fixed reference system as represented in the figure. The origin of the Cartesian system is the robot initial position, from where the entire process starts. Even though the location of the doors is indicated, the robot keeps checking for the door while approaching it, to calibrate its position and its state (open/closed, etc.). The semantic navigation framework is also used for completing other tasks, such as localizing a person or an object.

To gain planning stage flexibility, we also developed a way-points based navigation approach. In this way, to move the robot between two locations in the map, we can force it to follow intermediate points not belonging to a specific or optimal path. This is particularly useful to test motion features in specific parts of the scenario, or to reach specific places during the trajectory (e.g., to force the robot to pass through a specific door). The path between intermediate points is computed by the ROS path planner. A dictionary data structure is used to represent way-points paths: The keys are entity pairs (i.e., the source and the destination in the map), and the values are the list of places reached along the path. The way-points dictionary is stored as a *json* file. The way-points based navigation is activated if the pair source-destination is present in the dictionary.

2.4 The "Help Me Carry" Context

As a realistic scenario for door opening, we based our study on the "Help Me Carry" task included in *Robocup 2018*. To complete it, the robot has to memorize locations, move following user commands, avoid obstacles, and open doors. The task description is as follows. The user went shopping, and needs the robot's help for bringing inside all the bags. To complete the task the robot will:

1. Follow the owner to the bags.
2. Memorize the bags location.

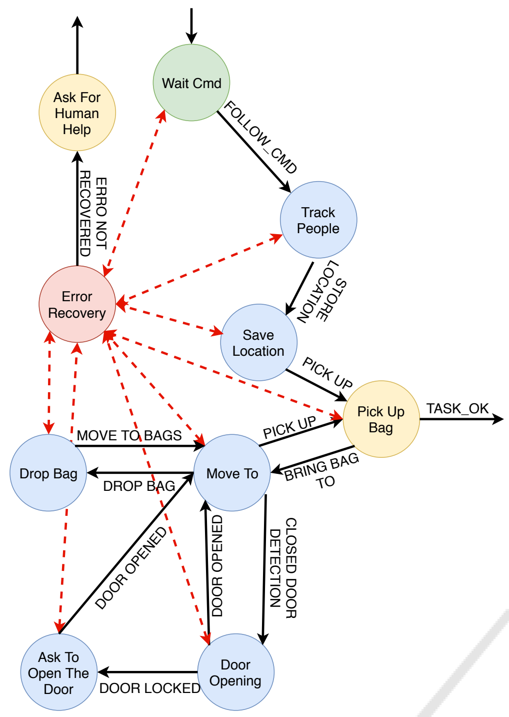


Figure 3: Automaton representing the "Help Me Carry" task. It shows the problem of door opening in the context of a more complex task, which involves human interaction and navigation.

3. Understand the owner's command to bring the bags to a specific different location.
4. Bring all bags to that desired specified location.

The automaton designed to perform the task is shown in Figure 3. Blue circles indicate operational states, green ones are initial states, and yellow ones represent ending states. The red color represents error recovery states. Black and red dashed arrows indicate transitions between states and transitions between a state and the error recovery state, respectively. The red lines are bi-directional because after the error handling the control may be given back to the calling state. The text on the arrows represent the event causing the transition. Each state is implemented as an automaton, hence the overall architecture is a hierarchical state machine. For the sake of readability, we did not used the often used "double border" notation to identify nested state machines. This structure is quite flexible and it is easy to maintain.

As an example of behavior, the robot is activated in the state named "Wait Cmd" (wait for command). In this state the robot simply waits for commands coming from the user. If the command for following the user is received, the state machine transit to the "Track People" state. Otherwise, if the command cannot be correctly interpreted, the state machine transits

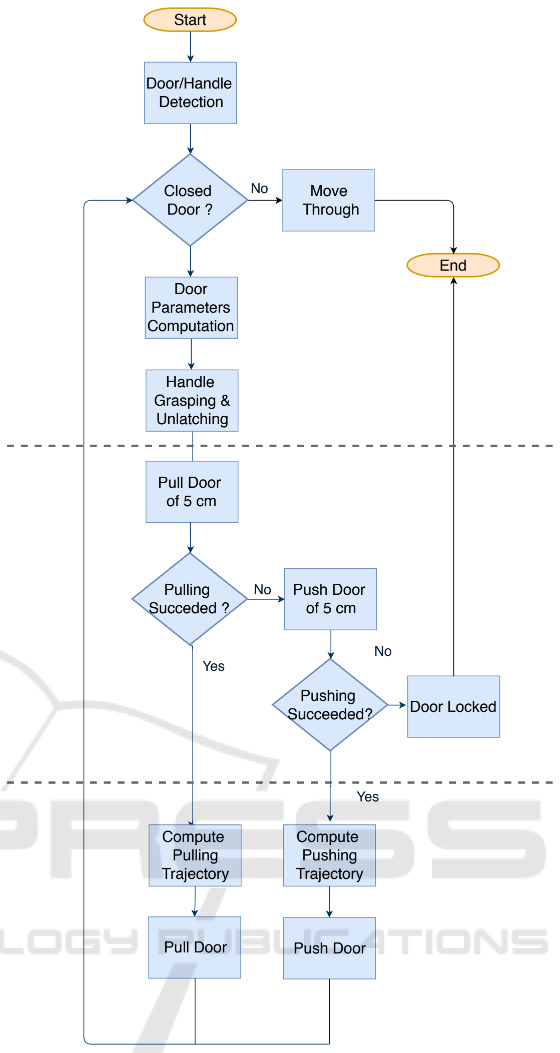


Figure 4: The operational flowchart for door opening. It includes the entire flow from the moment if which the robot detects a door to the one in which it crosses the door or it understands that the door is locked.

to the "Error Recovery" state. The general policy of the "Error Recovery" state is that, if the error is rectified, the control is given back to the incoming state. If the error cannot be rectified, the state returns the control to a higher level state machine or directly interacts with the user asking for help.

3 NESTING AUTOMATA

Detecting, opening and navigating through doors is a complex problem that involves many algorithms. In our approach, we decomposed the problem into different stages. The flowchart in Figure 4 describes the algorithmic approach we followed.

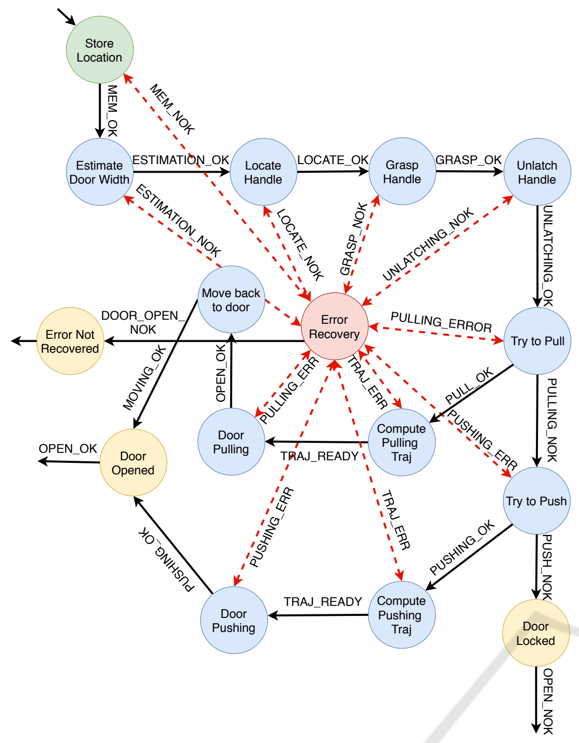


Figure 5: Our automaton for door opening. The names over the red dashed lines indicate the type of transition between a state and the "Error Recovery" state.

Each block involves different technologies and techniques. The top part represents the overall door/handle detection, and the door parameters estimation. The door type (pulling or pushing) is checked in the central part, whereas the opening phase is executed at the bottom part. In summary, the robot autonomously recognizes the door, it localizes the handle for grasping, and it decides the opening action (i.e., pulling or pushing). To open the door, the robot needs to know two parameters, i.e., the opening direction (pushing or pulling), and the door width. Following many other approaches, these characteristics could be annotated in advance in the environment description. However, we want to achieve a flexible and completely autonomous interaction with the door. Therefore, our robot computes the door width and the opening direction at run-time. The automaton implementing our door opening approach is shown in Figure 5. Notice that this state machine is nested in the automaton designed for the overall "Help Me Carry" task and previously described in Figure 3. The door opening state machine is launched when the robot detects a closed door. In the first state the current location is memorized. The following states complete the entire process described in the flowchart. The automaton has 3 ending states:

- "Door Opened": Reached when the door is open.
- "Door Locked": Reached if the door is locked.
- "Error Not Recovered": Reached if an error that prevents door opening occurs.

If the "Door Locked" or the "Error Not Recovered" states are reached, the door can not be opened. This situation is managed by the state machine working at a higher hierarchical level (i.e., the one in Figure 3). Our error recovery approach plays an essential role to reach robustness and flexibility against unexpected situations. First of all, the error is handled locally within the state in which occurs. For the sake of usability, the robot should not rely on human help for solving minor issues. Thus, in our framework, each state stores enough knowledge of the situation to handle minor problems. Examples of minor errors are: A wrong handle recognition in the 3D space, a grasping failure, a wrong location spelling from the user, etc. If local error correction is not possible, the control flow jumps to the previous (higher) hierarchical level, in which the error recovery state tries more drastic error rectification procedures. Only after the system has attempted all error recovery procedures, the robot will ask for help from the human operator.

3.1 Door and Handle Detection

For the door and handle detection we use a deep learning approach. Several deep neural networks have been proposed for object detection, and more specifically for door and handle recognition. Among state-of-the-art networks, we decided to exploit the Single Shot MultiBox Detector (SSD) neural network (Liu et al., 2016). Authors proved that this network outperforms other well know networks, like Yolo and Faster R-CNN in terms of speed and accuracy. Moreover, since SSD performs better on embedded systems, the network can work correctly at run-time, and it guarantees a fast interaction with the environment. Compared to other single shot methods, SSD provides a much better accuracy, even with a smaller input image size. The input to SSD is a monocular color image, and the output is a list of bounding boxes containing the detected objects in the image, namely, the top left angle of each detected object plus its height and width (*object detection* part). Each detected object has an associated label indicating which class the object belongs to (*object recognition* part).

In our version of SSD, the object recognition part is based on the VGG16 model pre-trained on the ILSVRC CLS-LOC data-set (Russakovsky et al., 2015). Then, we trained the object detection part, and fine-tuned the object recognition part, by constructing our own data-set, the "MIL-door" data-set. The "MIL-door" data-set consists of images of "doors"

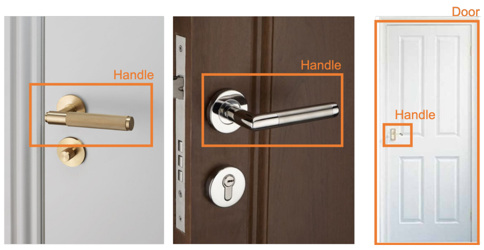


Figure 6: Sample images from the “MIL-door” data-set.

and “handles” crawled from Google Images. After filtering the erroneous results, MIL-door contains 462 images of doors and 318 images of handles, for a total of 780 images. The height and width of the images range from 400 to 1200 pixels. For each image, we manually annotated bounding boxes delimiting the area corresponding to doors and handles. Annotations are not inserted on top of the images, but stored in a separate text file. Figure 6 shows three example images extracted from our annotated data-set.

When training our SSD network with the MIL-door data-set, we performed data augmentation on the training data, namely, 90 degrees rotations and horizontal flips. This increases the size of our data-set eight times, for a total of 6240 images. Considering that the object detection part of the original SSD was trained with 9963 images for 20 object classes, we believe our data size is reasonable for our 2 object class detection problem.

As training parameters, we used the following configuration (please refer to (Liu et al., 2016) for more details on the meaning of these parameters): Batch size 32, maximum iterations $120,000$, learning rate 0.001 (the original learning rate is decayed by 10 at iterations $80,000$, $100,000$ and $120,000$), weight decay 0.0005, γ 0.1, momentum 0.9.

We used a low learning rate to assure convergence during training and we selected it empirically. We evaluated our door and handle detection with our MIL-door data-set using a 10-fold cross-validation setting. We consider that the door (or handle) has been correctly detected if the intersection over union (IoU) between the estimated bounding box and the annotation is greater than 85%. The detection accuracy in this controlled setting is of 94.7% for doors, and 86.3% for handles. However, during the evaluation in a real setting, the IoU recognition accuracy was slightly lower than using the data-set images. This was mainly due to three factors: The large diversity of doors that exist in the real world, the small size of some handles, and sporadic image quality loss due to poor lighting conditions.

Since there are cases in which the door is detected but the handle is not, we designed an error recovery algorithm to add robustness. When a door is detected but the handle is not, the robot moves slightly forward, backwards, and laterally to change the perspective until the recognition succeeds. If the handle is not detected after a certain number of trials (5 in our case), the error is passed to the above error recovery state in the state machine hierarchy.

3.2 Door Width Computation

The door width is an important parameter to correctly estimate the robot’s trajectory. To compute it, we combine the door size in the image, taken from the robot camera, and the door to robot distance computed using the depth camera. Assuming that the object width on the image is $width_{image}$, and the detected distance is d, we can obtain the relative size in the real world using the following formula:

$$width_{real\ [pixel]} = width_{image} \cdot d. \tag{1}$$

However, Equation 1 measures the door size using the pixels as measurement unit. To transform the computed value from pixel into centimeters, we empirically calibrated our camera and we computed a conversion factor $conversion_{coeff}$. The door width, expressed in length units (centimeters), is thus given by:

$$width_{real\ [cm]} = d \cdot conversion_{coeff} \cdot width_{image}. \tag{2}$$

We measured the quality of our method by comparing our estimated widths against ground truth values, on four different types of doors. These doors differ in terms of color, surface material, and shape. We also varied the distance of the robot from the door from 1 m to 3 m, measures that are somehow reasonable in a home environment. We used the root mean square error to evaluate the error. Our results show that we reached an average error of $\pm$ 6 cm. As observed in our experiments, this value does not affect the door opening noticeably.

3.3 Opening Direction

To open the door, the robot should move backward from left to right if the hinges are on the right, and vice-versa. Anyway recognizing the hinge position is not robust enough, since hinges are often undefined or barely visible. However, our handle and door detector provides the handle location with respect to the door, and thus, inferring the opening direction is straightforward. The opening direction is used to compute the opening trajectory for both pulling and pushing doors (see Sections 3.7 and 3.8, respectively).

3.4 Closed Door

The door detected in the door recognition phase may be already open. To check this, we use the HSR's RGB-D sensor, the Xtion PRO LIVE. First, we obtain the depth image corresponding to the frame where the door has been located. Then, we take two horizontal rows (e.g., one in the lower half and one in the upper). Finally, we compute the Sobel derivative along the horizontal direction of these lines, and we check if it contains values above a certain threshold t. This allows our method to detect if there are edges where the depth suddenly increases, which translates into the door being open.

We experimentally established that the door can be considered open if the log_{10} of the derivatives exceed a threshold $t = 3.5$.

3.5 Handle Grasping and Unlatching

Once the door opening direction has been established, and the distance from the door d evaluated, the robot can approach the handle enough to get a more precise measure of its location with the depth sensor. If some error occurs while evaluating the handle position, we retrieve a new depth measurement from the sensor to get the right location. The robot, with its grip open, gets in front of the door, and when it reaches the handle location, the grip closes and the robot grasps the handle. To unlatch the handle, we combine the robot hand rotation with a downward movement. We rotate the hand 20 degrees, and we move it downwards 10 *cm*. We empirically found that HSR does not have a strong grip and a rotation plus a downward movement can improve the pressure that the hand can apply to the handle. This allows a robust unlatching even if the handle is not grasped perfectly at its end, or the surface of the handle is slippery (e.g., metallic).

3.6 Door Type Checking

Before computing the opening trajectory the robot has to understand the door type, i.e., whether the door is a pulling or a pushing door. To discriminate between the two categories, after the grasping and the unlatching, the robot tries to move backwards and forward to test the opening type. First, it attempts to pull the door back 5 *cm* while monitoring the force acting on the wrist torque sensor. The measure of 5 *cm* has been heuristically selected as a good compromise among several requirements. If during this movement, the torque on the wrist sensor grows continuously, the door cannot be pulled. In this case, the HSR attempts to push the door by moving forward and it checks the force acting on the wrist sensor as before. In case the torque force does not increase in one of these two attempts, the robots start the opening phase (see Sections 3.7 and 3.8). On the other hand, if the door cannot be pulled or pushed, the robot assumes that the door is locked. The "Error Recovery" state handles this case by calling for human help.

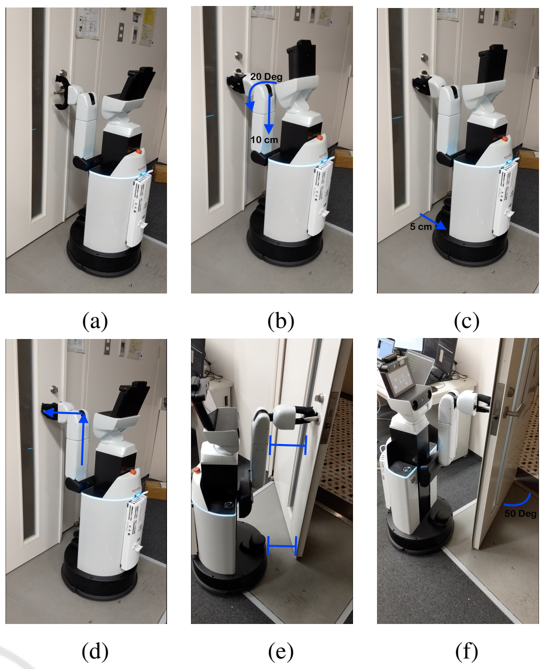


Figure 7: A visual example of our door opening approach. The HSR (a) grasps the handle and (b) unlatches it, then (c) tries to move back for 5 *cm* to pull the door. If the door cannot be pulled, the robot (d) moves the handle back to its neutral position, and (e) the door is opened by moving backwards and drawing an angle with respect to the door closing position. During the entire process (Figure (f)) the door-to-robot distance is maintained constant.

We also considered other approaches for testing the door type. One of those involves monitoring movement of the robot's base while performing the test. This approach did not succeed mainly because, to measure a significant movement of the base, we have to move the robot more than 5 *cm*. This in turn can damage both the robot and the door (e.g., by pulling a pushing door too hard). Another approach implies the classification of the door type using a computer vision approach. However, this solution depends largely on the size of the training data-set, which should contain a wide variety of doors and annotations indicating their type. Unfortunately, many available images are not annotated, and manually create a large data-set is very time consuming.

Notice that all checks performed by our approach are done to assure robustness and to minimize the number of errors. We emphasize the importance of

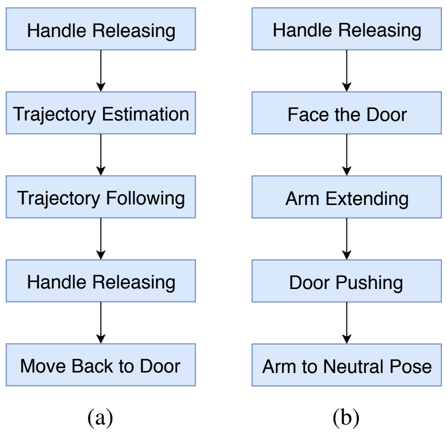


Figure 8: Schematic code flow for: (a) opening a pulling door, and (b) opening a pushing door. The code flows are encoded as SMACH state machines, and they are fully integrated in our software framework.

robustness in such a complicated scenario, since an error in door type recognition could lead to hard-to-manage situations or risks for the robot or the handle and the door integrity.

3.7 Door Pulling

Figure 7 shows the entire flow for opening a pulling door, from the moment the robot must grasp the handle to the one in which the door is open. Figure 8a shows the corresponding code flow.

When the robot stands in front of the door, and before starting the door pulling phase, the application stores the current robot position. These coordinates will be used when the door is open, as the robot will move back to the stored position to pass through the door. The first three images (Figure 7a, 7b and 7c) are part of the door type understanding process described in Section 3.6. In the latter phase, the robots moves backward 5 *cm* to check whether the door is a pulling one. In the affirmative case, the robot moves the handle back to its neutral position. A visual representation is given in Figure 7d. This action emulates typical human behavior, and it effectively reduces the load on the robot wrist that does not need to hold the handle down. At this point, the robot computes the pulling trajectory as shown in the second block of Figure 8a. The final trajectory is an arc-shaped sequence of map coordinates that form an angle of 80 degrees with respect to the door hinges. In this way, the door is opened wide enough for the robot to pass through it.

Because the HSR's arm has less than six degrees-of-freedom (DoF), we have to move the base and the arm together, keeping the robot hand in a fixed position. As a consequence, the door-to-robot distance remains constant. In this way, we do not need to continuously check for collision between the robot and the door. This situation is shown in Figure 7e. Once the robot completes the trajectory, it releases the handle, and it moves back in front of the door to continue the navigation toward the final goal. The robot position saved in the first state is used as a target position to cross the door.

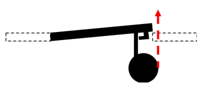
3.8 Door Pushing

Following Figure 4, if the robot detects that the door cannot be pulled, it checks whether it can be pushed, and, in this latter case, the pushing process starts. The pushing door action flow is detailed in Figure 8b. As in the pulling door case, our robot attempts to push the door to check the opening type. After the handle releasing phase, the robot moves in front of the door at a fixed distance of 50 *cm*. Once this position is reached, the robot first extends its arm to reach the door, which is already open a few centimeters after pushing it to check its type. As the robot is going to move forward, reaching the door is not strictly necessary. At the same time, we also monitor the wrist sensor to assure that no unexpected collision occurs. During the pushing phase, the HSR moves forward, and when the phase finishes, the robot is on the other side of the door. The last action executed by the robot before restarting the normal navigation, is to retract its arm into its original and safer position.

To succeed in the pushing action, the handle position is an important parameter. When unlatching the handle, the robot faces it, but during the pushing action, some collisions may occur. Since HSR is a left-handed robot, the most unfavorable scenario is when the handle is on the right side. A schematic top-view of this situation is given in Figure 9. While pushing the door, a collision check is performed in the robot base to prevent HSR from hitting the door frame. If a potential collision situation is detected, the robot is moved slightly to the left with respect to the handle. If a collision is detected, the "Error Recovery" state stops the robot and moves it back to the beginning of the pushing stage. These strategies were validated empirically, and allowed for a safe and robust navigation through doors, as described in the next section.

4 EXPERIMENTAL ANALYSIS

We evaluated our unified framework by means of two set of experiments. These experiments were designed to verify two main aspects: 1) Our framework's ro-



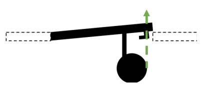


Figure 9: Passing through a pushing door: (a) HSR may suffer a collision when opening a pushing door with a right side handle. (b) To avoid hitting the door frame, the sensor on the robot base is activated. If HSR detects a possible collision, its position is slightly shifted to the left.

bustness in a real navigation scenario, and 2) The quality of the entire door opening process with different doors, handles, materials, etc.

First, we evaluated the door opening process in a realistic navigation scenario by using a simplified version of the "Help Me Carry" task previously described. In this task, the user instructs the robot to fetch an object in a specific location in a different room, and he awaits for the robot to return. We also imposed way-points during navigation, i.e., we force the robot to follow a different path on the way back. To run this scenario, we arranged a house environment similar to the one in Figure 2. Initially, the HSR robot is in a location within room R1. The robot is supposed to reach room R4 by passing through doors D1, D2, and D4. Then, it should go back to the initial position by passing through doors D3 and D1. The doors in this task have different characteristics. When moving from R1 to R2, door D1 is a pushing door with the handle on the left. Door D2 is open. when moving from R5 to R4, and door D4 is a pulling door with its handle on the left. On the way back, when moving from R4 to R2, door D3 is a pulling door with its handle on the left. Finally, when the robot moves back from R2 to R1, D1 is still open. The robot detected the doors during navigation, following a route determined by the ROS path planner. Since the experiment does not involve any obstacles, we did not employed the way-points navigation approach. Notice that the door type and handle position affects the door opening process in terms of the selected trajectories and the final success rate. In order to show the robustness of our framework, the door and handle attributes are unknown by the robot.

We commanded the robot to execute the task 50 times. In all cases, the robot reached R4 without navigation errors, and it successfully detected and discriminated between closed and opened doors. The accuracy of the door and the handle detection in the real scenario does not vary significantly with respect to the detection accuracy reported for our MIL-door data-set. Whenever a handle was not initially recognized, the error recovery procedure forced the robot to move slightly forward, backwards, and laterally to change the perspective until the recognition was successful. This procedure provided a recognition success rate up to 95%. In the remaining 5%, the error persisted so the higher hierarchical automata level dealt with it. Moreover, even if initially the location of the detected handle was not aligned perfectly, the location was refined when approaching the handle and using depth images. Regarding the handle grasping, every time the HSR could not hold the grip on a handle, the error recovery procedure reactivated the detection phase and the "door opening" phase restarted from the beginning.

In light of these results, we designed a second experiment with an emphasis on the handle grasping sub-task. In this experiment, the HSR had to deal with a variety of doors and handles, which differ in terms of door type (pushing or pulling), handle position (left or right), and material (slippery or non-slippery). We commanded the robot to move from room R1 to room R2 while modifying the configuration of D1. The robot starts in front of the door ready to grasp the handle, and it stops after the door is open (passing through is not required). As above, the robot does not know the door and handle attributes. We conducted 20 runs for each door and handle configuration. Notice that the door type influences the robot trajectory, whereas the handle material influences the quality of the handle grasp and its holding process. Moreover, some metallic handles may cause noise in the depth image due to reflections. We separate the door opening results for slippery handles (metallic), and non-slippery handles (wood or plastic-like material), and their location with respect to the door (i.e., left or right). Similarly, we also consider spring loaded doors, that is, doors that close by themselves after they are open. We do not evaluate opening pushing spring loaded doors since, once the robot arm releases the handle after the unlatching, the door closes again before the HSR has the chance to push it.

Table 1 summarizes the results for this second experiment. The handle localization using depth images proved to be robust with different handle shapes and materials. After the handle grasping, our approach recognized in 100% of the cases the door type, i.e., whether the HSR had to pull or push the door. As the HSR grip did not have enough strength to hold slippery handles (in particular, those in spring loaded doors) the door opening did not always succeed. However, when an error arose, the robot was able to retry the task by itself by following the error recovery procedure previously described. The robot asked for human help only in a total of 3 occasions. This results are very promising for a practical application, as the recovery procedure is able to rectify errors in most cases. However, for the sake of fairness, Table 1 considers runs as failed whenever an error arose, even if the robot recovered from the error autonomously. Overall, we reached a 98% of success rate for non-slippery handles, and 94% for slippery metal-like handles. Notice that these results are influenced not only by the robot's grasping ability, but also by the handle detection under different types of light reflection on the handle surface. Regarding pulling spring loaded doors, holding the handle when opening was quite challenging for the robot, specially in the case of slippery handles. This is due to the limited strength of the HSR's grip. Moreover, handles on the right side of pushing doors are more challenging due to the reasons explained in Section 3.8.

Table 1: Results of our door opening approach. The table presents the number of successes out of 20 opening attempts, with 4 different handle types. T_1: Slippery handle on the door left side. T_2: Slippery handle on the door right side. T_3: Non-slippery handle on the door left side. T_4: Non-slippery handle on the door right side.

Action Type	Handle Type			
	T_1	T_2	T_3	T_4
Pulling non-spring loaded door	16	18	18	19
Pulling spring loaded door	16	18	19	19
Pushing non-spring loaded door	20	18	20	17

5 CONCLUSIONS

In this paper we presented a unified framework for approaching, opening, and navigating through doors. To the best of our knowledge, this is the first attempt to solving the door opening problem in a navigation scenario. Our unified framework integrates an automata model and its state machine hierarchy. This includes techniques for error recovery, allowing for a robust door opening and its implementation in an operational system. We implemented our framework on a Toyota HSR, which is a standard platform and, thus, it facilitates the reproducibility of our work. For door and handle detection, we proposed a deep learning-based method trained with our image dataset. For handle grasping, door type checking, unlatching and opening, we propose optimized techniques for HSR, but extrapolable to similar off-the-shelf platforms. From the software engineering point of view, this paper covers the design, analysis and synthesis of such a robotic system for real-world operation. We evaluated our framework for navigation and door-opening approach in a challenging realistic scenario inspired by *Robocup 2018* tasks. We tested our platform against different types of doors, and with different types of handles and opening directions. Our results show the robustness and flexibility of our approach and its high applicability by using a standard service robot. On the other hand, robots lacking some of the HSR features (e.g., a depth camera, a base sensor, a wrist torque sensor) may not be able to implement our framework without previously adapting the algorithms.

As future work, we plan to attempt recognizing and opening a wider variety of doors. We will also extend our framework for recognizing and moving obstacles during navigation. We believe this can further improve the robustness and the flexibility of our framework against changes in the environment.

ACKNOWLEDGMENTS

We would like to thank Yusuke Kurose, Yujin Tang, Jen-Yen Chang, James Borg, Takayoshi Takayanagi, Yingy Wen and Reza Motallebi for their help implementing this research. This work was partially supported by JST CREST Grant Number JPMJCR1403, Japan. The authors have been part of the HSR developer community[10], and they made use of HSR hardware and software platforms.

REFERENCES

Andreopoulos, J. A. and Tsotsos, J. K. (2007). A framework for door localization and door opening using a robotic wheelchair for people living with mobility impairments. In *Robotics: Science and systems, Workshop: Robot manipulation: Sensing and adapting to the real world, Atlanta*.

Aude, E. P., Lopes, E. P., Aguiar, C. S., and Martins, M. F. (2006). Door crossing and state identification using robotic vision. *IFAC Proceedings Volumes*, 39(15):659 – 664.

Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., and Thrun, S. (1998). The interactive museum tour-guide robot. In *Proc. Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, pages 11–18.

Chitta, S., Cohen, B., and Likhachev, M. (2010). Planning for autonomous door opening with a mobile manipulator. In *International Conference on Robotics and Automation*, pages 1799–1806.

Dongwon, K., Ju-Hyun, K., Chang-Soon, H., and Gwi-Tae, P. (2004). Mobile robot for door opening in a house. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 596–602.

Fernández-Caramés, C., Moreno, V., Curto, B., Rodríguez-Aragón, J. F., and Serrano, F. (2014). A real-time door detection system for domestic robotic navigation.

[10]https://newsroom.toyota.co.jp/jp/detail/8709536

Journal of Intelligent & Robotic Systems, 76(1):119–136.

Gray, S., Chitta, S., Kumar, V., and Likhachev, M. (2013). A single planner for a composite task of approaching, opening and navigating through non-spring and spring-loaded doors. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3839–3846.

Hernandez, K., Bacca, B., and Posso, B. (2017). Multi-goal path planning autonomous system for picking up and delivery tasks in mobile robotics. *IEEE Latin America Transactions*, 15(2):232–238.

Jain, A. and Kemp, C. C. (2008). Behaviors for robust door opening and doorway traversal with a force-sensing mobile manipulator. In *RSS Workshop on Robot Manipulation: Intelligence in Human Environments*.

Johnson, M., Shrewsbury, B., Bertrand, S., Wu, T., Duran, D., Floyd, M., Abeles, P., Stephen, D., Mertins, N., Lesman, A., Carff, J., Rifenburgh, W., Kaveti, P., Straatman, W., Smith, J., Griffioen, M., Layton, B., Boer, T., Koolen, T., Neuhaus, P., and Pratt, J. (2015). Team ihmc's lessons learned from the darpa robotics challenge trials. *J. Field Robot.*, 32(2):192–208.

Khatib, O. (1999). Mobile manipulation: The robotic assistant. *Robotics and Autonomous Systems*, 26(2):175 – 183.

Kim, G., Chung, W., Kim, K.-R., Kim, M., Han, S., and Shinn, R. (2004). The autonomous tour-guide robot jinny. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 4, pages 3450–3455.

Kim, S., Cheong, H., Kim, D. H., and Park, S. (2011). Context-based object recognition for door detection. In *15th International Conference on Advanced Robotics (ICAR)*, pages 155–160.

Klingbeil, E., Saxena, A., and Ng, A. Y. (2010). Learning to open new doors. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2751–2757.

Kohlbrecher, S., von Stryk, O., Meyer, J., and Klingauf, U. (2011). A flexible and scalable slam system with full 3d motion estimation. In *IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 155–160.

Krotkov, E., Hackett, D., Jackel, L., Perschbacher, M., Pippine, J., Strauss, J., Pratt, G., and Orlowski, C. (2017). The darpa robotics challenge finals: Results and perspectives. *J. Field Robot.*, 34(2):229–240.

Lim, J., Bae, H., Oh, J., Lee, I., Shim, I., Jung, H., Joe, H.-M., Sim, O., Jung, T., Shin, S., Joo, K., Kim, M., Lee, K., Bok, Y., Choi, D.-G., Cho, B., Kim, S., Heo, J., Kim, I., and Oh, J.-H. (2018). *Robot system of DRC-HUBO+ and control strategy of team KAIST in DARPA robotics challenge finals*, pages 27–69.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). SSD: Single shot multibox detector. In *European conference on computer vision*, pages 21–37.

Meeussen, W., Wise, M., Glaser, S., Chitta, S., McGann, C., Mihelich, P., Marder-Eppstein, E., Muja, M., Eruhimov, V., Foote, T., Hsu, J. M., Rusu, R. B., Marthi, B., Bradski, G. R., Konolige, K., Gerkey, B. P., and Berger, E. (2010). Autonomous door opening and plugging in with a personal robot. In *IEEE International Conference on Robotics and Automation*, pages 729–736.

Ott, C., Bäuml, B., Borst, C., and Hirzinger, G. (2007). Autonomous opening of a door with a mobile manipulator: A case study. *IFAC Proceedings Volumes*, 40(15):349–354.

Peterson, L., Austin, D., and Kragic, D. (2000). High-level control of a mobile manipulator for door opening. In *Proceedings. IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2333–2338.

Petrovskaya, A. and Ng, A. Y. (2007). Probabilistic mobile manipulation in dynamic environments, with application to opening doors. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2178–2184, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Rhee, C., Chung, W., Kim, M., Shim, Y., and Lee, H. (2004). Door opening control using the multi-fingered robotic hand for the indoor service robot. In *Proc. IEEE International Conference on Robotics and Automation*, volume 4, pages 4011–4016.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252.

Rusu, R. B., Meeussen, W., Chitta, S., and Beetz, M. (2009). Laser-based perception for door and handle identification. In *2009 International Conference on Advanced Robotics*, pages 1–8.

Shalaby, M. M., Salem, M. A., Khamis, A., and Melgani, F. (2014). Geometric model for vision-based door detection. In *9th International Conference on Computer Engineering Systems*, pages 41–46.

Thrun, S., Bennewitz, M., Burgard, W., Cremers, A. B., Dellaert, F., Fox, D., Hahnel, D., Rosenberg, C., Roy, N., Schulte, J., and Schulz, D. (1999). Minerva: a second-generation museum tour-guide robot. In *Proc. IEEE International Conference on Robotics and Automation*, volume 3, pages 1999–2005.