




# LEVY Gabriel

Student at PHELMA


Internship report
Septembre 2022 - March 2023

## Opening a door with a robot


tutors:

Peter van Dooren: PhD student
René van de Molengraft: Associate professor

# Contents

# List of Figures

# Introduction

Robot are more and more used nowadays and they have to be in environments that are becoming more and more complex. To work in such environments, they must be able to perform high level tasks. According to Wikipedia, a high level task is a task that is more abstract in nature wherein the overall goals and systemic features are typically more concerned by the system as a whole [2]. To perform a high level task, the robot needs to get data from the environments, analyze them and pick its actions according to the results. Most of time, a high level task can be subdivided in primitive task that can be realized by primitive actions. A primitive action can be considered as grounded robot skills that can be directly implemented on a robot system [1].

The objective of this document is to explain the implementation of a high level task: **opening a door**. The task was performed with Hero, a robot from the at_home team from Tech United. For the communication, the framework ROS has been used and the code was written in Python and C++. In Python, the SMACH library allows to break down a high level task in different primitive tasks by calling a task planner after each action execution. The document is made for people who want to use or improve the door opening with Hero. It may also help to understand the general approach of a high level task.

The paper is divided in six parts as it is shown in the content. The first part: procedure to open the door describes and explains the commands to open the door with the robot Hero while the second part describes the repartition of the files that contain the code. The part about general approach details the reasonning of two complex functions: the detection of the handle and the door pulling. The part named technical details explains the implementation and the tools used in the service server,in the robot movement in front of the door and in the door passing. The fifth part introduces some limits in the work while the last part is about future work.

# 1 Procedure to open the door

There are two computers on the robot Hero, the first one is inside the hardware and it is named hero1. The second is the laptop on the back of Hero and it is named hero2.

## 1.1 Go on the appropriate branch

All the branch are on the at_home git repository. The code to open the door is on the branch **gabriel**, so, when it is asked to switch branch, the destination branch is named **gabriel**. The command to switch is: **git checkout gabriel**. To go to a specific folder in the software, the command is **roscd <name of the folder>**.

### 1.1.1 First computer: hero1

There is one folder that must switch branch on hero1, it is named **hero_bringup**.

### 1.1.2 Second computer: hero2

There are four folders that must switch branch on hero2, they are named:

**hero_bringup**
**robot_smach_states**
**ed_object_models**
**root_launch_file**

## 1.2 Commands to run

These commands are the one that open the door. They must be executed after the switching of the branchs.

### 1.2.1 Step one: on hero1

On hero1, the command is **sudo systemctl restart hero1-start.service**. After the execution of the command, the user needs to restart and kill all the rosnode on Hero by switching it off-on three times in a row.

### 1.2.2 Step two: on hero2

On hero2, several commands need to be executed. All of them are commands that must keep running while the main code is in execution. It means the user needs to have one dedicated terminal for every commands. They are:

**hero-free-mode**
**rosrun robot_smach_state open_the_door_service**
**roslaunch laser_line_extraction example.launch**

#### 1.2.3 Step three: on the user's computer

Then the robot must be positionned in its environment in RVIZ and it can not be done in hero1 or hero2. It must be done on the user's computer that must contains the at_home software. He must first open a terminal, then write the command **hero-core**, then **hero-rviz** and then give the most accurate position he can to the robot in RVIZ.

#### 1.2.4 Step four: on hero2

The last command, is the one to run the code that open the door. It is:

**rosrun robot_smach_state open_door_smach.py**

After the execution of all of the commands, the robot should move in order to open the door.

# 2 Files specific to the door opening

## 2.1 World model

In order to know the world, hero needs a world model that defines and positions the objects of the laboratory Impuls. It must be done in a **Standart Data File** (SDF). For Hero, the world model is writen in the folder **ed_object_model** (command **roscd ed_object_model** to go there) and the Impuls description is in **models/impuls/**. The file **models/impuls/model.sdf** decribes the components, their positions and orientations in Impuls. In this folder, the models of the door are written on the branch (related to git tools) **gabriel**. There are two additional folders on this branch: **models/impuls/door_inside** and **models/impuls/door_inside_v2**.

We will use the concept of virtual volume. A virtual volume is a volume that is present only in the world model. It is writen in the model file, and it allows to get a fixed location in the map. In the model there are five virtual volumes that are named **door_vv**, **handle_vv**, **handle_behind_vv**, **frame_left_point_vv** and **frame_right_point_vv**.

## 2.2 Command the robot

As said earlier, the code is written in both C++ and Python . All the code written to open the door is in the folder **robot_smach_states** (use command **roscd robot_smach_states** to go there). In the folder, the code is written on the branch (related to git tools) named **gabriel**. The files that are modified in respect with the branch **master** are **CmakeLists.txt**, **package.xml**. The files that have been created in the branch **gabriel** are **srv/door_info.srv**, **src/robot_smach_states/manipulation/service_server.cpp** and **src/robot_smach_states/manipulation/open_door_smach.py**.

**door_info.srv** is the file describing the message that is used by the service server, **service_server.cpp** is the file that is used to launch and handle the service server and **open_door_smach.py** is the file that implement the main code that is used to open the door.

### 2.3 Library laser_line_extraction

The laser_line_extraction library is a library availables on gitHub. The library allows to analyze the data from the laser that is present on hero and to extract segments from them. It is stores in the catkin workspace of hero2 (command **trunk** in a terminal to go there) and it is not stored on the at_home git repository.

The difference with the repository available on gitHub is the file **laser_line_extraction/launch/example.launch** that has been modified in order to operate with Hero. The file is written is the appendix B.

## 3 General approach

### 3.1 Detection of the handle

The detection of the handle is a key part of the door opening, the position of the handle could have been hard coded but in order to make the code more robust, a detection based on a RGBD sensor was done. The detection of the handle is made by the service server in 5 steps:

The first step is getting data from the RGBD sensor

The second step is cropping the data received around the handle's virtual volume. To do this, a cube with an edge measuring 0.5 meter is created around the center of the virtual volume and all the data that are outside the box are removed. Figure 1 represents the door, the handle, the virtual volume and the box in two dimensions .

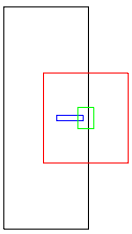


Figure 1: *The door is black, the handle whose locaion is to be dtermined is blue, the virtual volume is green and the box around it is red.*

All the data that were transmitted by the RGBD sensor that are not in the red box are removed. It allows to select the interesting part.

The third step consists in removing large plane surfaces from the data in order to reduce the number of points and as only the biggest plane surfaces are removed, it is not likely that the handle's points would be remove.

Then, the fourth step is extracting clusters from the data. A cluster is a group of data that has similarity and it is very likely that one of these cluster is the one representing the handle. In order to not have too many cluster, the third and fourth step are applied in a while loop whose breaking criterion is having less than ten clusters. As long as there are more than ten clusters, a plane surface is removed.

After the extraction of clusters from the data, the fifth and last step is to perform a selection on the clusters in order to get the one representing the handle. The first criterion of selection is the global distance to the virtual volume, the center of the cluster has to be in less than 0.3 meter from the center of the virtual volume. Figure 2 illustrates the critiria in two dimensions. All the cluster that are outside the circle are eliminated. It allows to select only

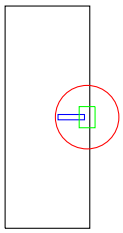


Figure 2: *The door is black, the handle is blue, the virtual volume is green and the circle around the virtual volume is red. It represents the maximum distance a cluster can be from the virtual volume.*

the one that could be the handle. The other one are too far away.

Then a criterion on the position of the cluster comparated to the position of the virtual volume is applied. A vertical line that goes through the middle of the virtual is drawn The cluster must be on the good side of the line as it is shown figure 3.

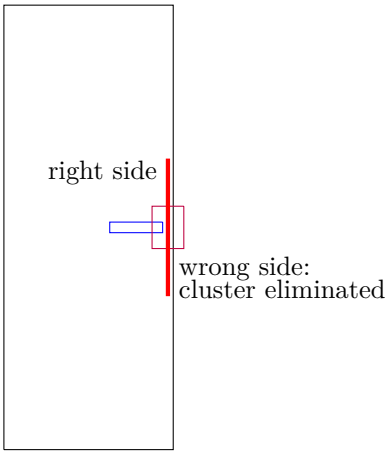



Figure 3: *The door is black, the handle is blue, the virtual volume is purple and the line separating wrond and good side is red.*

If the clusters are not on the good side of the virtual volume, they are not susceptible to represent the handle, so they are removed from the possibility.

At the end of these two selections, there are usually a few clusters remaining. Among the remaining ones, the cluster closest to the robot is choosen, because the handle is more proeminent than the door. The figure 4 is two screenshots from the simulation that shows the clusters. Only three clusters were detected and the one representing the door is indeed in front of the other two. In the

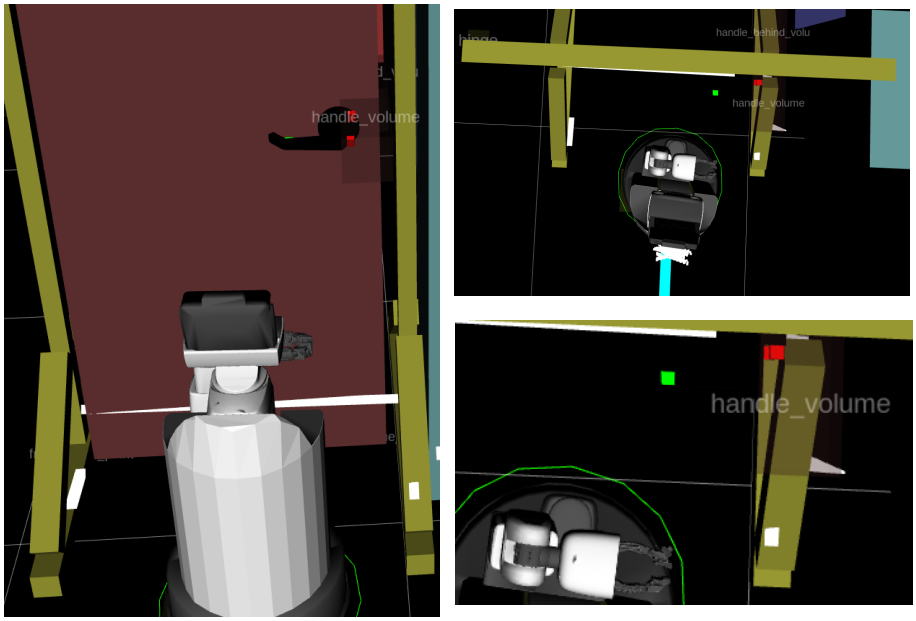



Figure 4: *The three images are screenshots from RVIZ, the ros visualization tool. The door is purple, the door frame is yellow and Hero is white. The virtual volume* ***handle_vv*** *is drawn and is named* ***handle_volume****. The left image is a view from the front while the top right is a top view. The bottom right is a zoom from the top right. The three color cubes represent the clusters. The green one is the choosen one and the two red cubes represent eliminated clusters.*

figure 4 applying only the third selection would have been enough. Nevertheless, the frame of the door or the wall next to it could contains clusters. It is why the other selections are important.

The proper cluster is then used by the grasping function as the point to target in order to grab the handle. The detection of the handle is a major part of the door opening, it needs to work well. A series of fifty tests was performed on the door in the laboratory impuls and, in these test, the detection had a probability of success of 0.95.

## 3.2 Pulling the door

Depending on the orientation of the door, the door needs to be pushed or pulled, to be opened. Pushing or pulling the door occurs after the the handle has been unlatched. When pulling the door, an additional difficulty arises: the robot must keep the handle in its gripper, in order to pull the door while moving backwards

Pulling the door is made using the laser_line_extraction library. The library analyses the laser data to find segments in it. Each segment is described by few information: a **radius**, an **angle**, a **covariance** and two points: **start** and **end** that represent the beginning and the end of the segment.

Before starting to pull the door, the angle of the robot according to the door

and the distance between the robot and the middle of the door must by acquired and stored. Then, the pulling the door is done step by step.

The first step is to move the robot 10 cm back while the robot grips the handle and pulls the door. Moving back the robot is changing its angle according to the door, the robot also moves away from the middle of the door.

The second step is to rotate the robot until the current angle between the robot and the door is the same as the initial angle.

The third step is to translate the robot parallel to the door until the distance between the robot and the middle of the door is the same as the initial distance. Once the three steps are done, the position of the robot relative to the door is identical to the initial position, and the robot can perform the three steps again, to continue to pull the door.

The figure 5 shows the four positions the robot has when it pulls the door the door. After the first position, the robot is pulling the door in order to move to the second position. Then it rotates to the third position, and translates to the fourth and last position, which is identical to the first position relative to the door.

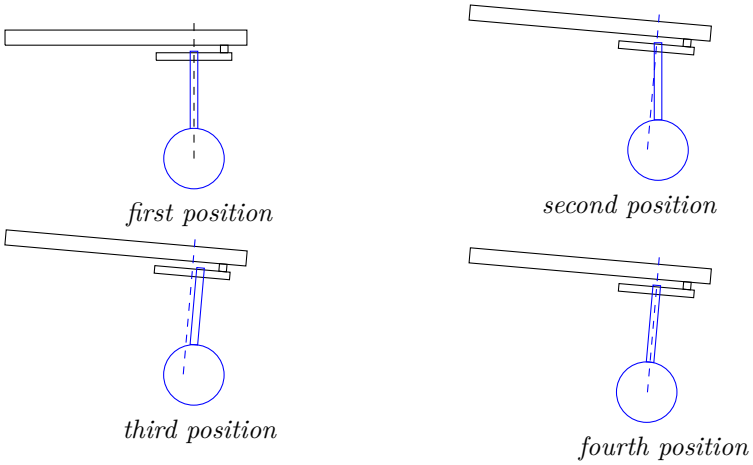



Figure 5: *The successives positions while the door is pulled*

When performing the tests, it appeared that the handle leaves the gripper in many cases. To overcome the situation, a maximum threshold to the number of rotations and to the number of translations parallel to the door has been applied. Each of them can not be done more than three times for one pulling the door. The threshold value was determined by experimenting several values.

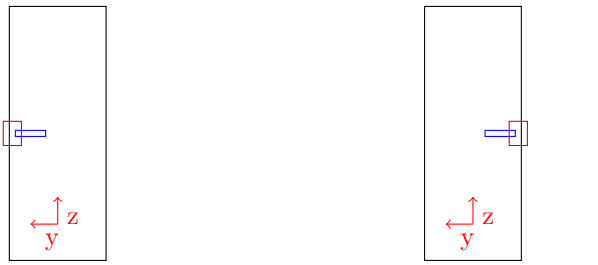
The pulling the door is a major part of the door opening, and even if it is success around 80% of the time, thare are still some problems. One of them is the time needed to pull the door. In impuls, Hero needed around one minute and 20 secondes to pull the door and the door was still not fully open. The time needed by Hero to go back then in front of the door has also to be taken into account.

# 4 Technical details

## 4.1 Service server

The service server is implemented by the file **service_server.cpp** and the type of message use to communicate with the server is described in the file **door_info.srv**. To send a request to the the server, the request must contain two fields: a **string**: **input_string** and a **geometry_msgs/PointStamped**: **point_in**. The server sends back a **geometry_msgs/PointStamped**: **point_out**. Three requests have a function on the server, they are differentiated by **input_string** whose three values can be **set_y_direction**, **write_marker** and **publish_marker**.

The request **set_y_direction**, is made to give to the server a direction according to the door reference system, the direction is either positive or negative according to the vector **y** of the door reference system. The sign of the direction is used during the second cluster selection, it allows to know which side of the virtual volume is the good. In the following explanation, the coordinate **y** of the cluster in the door reference system is named **y_cluster** and the coordinate **y** of the virtual volume in the door reference system is named **y_vv**. If the direction is positive, **y_cluster** must be superior to **y_vv** and if the direction is negative, **y_cluster** must be inferior to **y_vv** (figure 6).




*handle on the left* *handle on the right*

Figure 6: *The two positions a handle can have. Coordinates are expressed in the door referent (drawn in red).*

In the case with the handle on the left, the direction must be negative and in the case with the handle on the right, the direction must be positive. The sign of the direction is given using **point_in** that must be in the door reference system. Once the request has been done, nothing is written in **point_out**.

The request **write_marker** is done in order to detect the handle. The variable **point_in** is the point at the center of the handle's virtual volume in the map reference system. Once the request has been done, the variable **point_out** should be the point that is the center of the cluster that represent the handle, the coordinate of the point are in the frame of the robot.

The request **publish_marker** is done in order to display in RVIZ the center of all the cluster that were found during the handle detection. The selected cluster appears in green as it was showed in the figure 4. The value of **point_in** is not important when the request is done and it does not write anything in **point_out**. When the request **publish_marker** is done, the robot stop moving after, so it should be done only to work on the detection.

## 4.2 Moving front of the door or the handle

### 4.2.1 How does the robot move there

Before crossing the door, the robot needs to move to the door to evaluate the door state (either closed, open or intermediate) and later it also needs to go in front of the handle to detect it. To move there, the global planner present in the at_home software is used. To be executed, the global planner needs a location point and an orientation for the robot. The positions and the orientations are written in the class door, thay are hard coded according to the door reference system. By doing this, it allows to still have the right value if the door is moved in the real life and in the world model. So, the class door contains four functions that send back positions: **getFrameIFOdoor_face**, **getFrameIFOdoor_behind**, **getFrameIFOhandle_face**, and **getFrameIFOhandle_behind**. The results send back by the four functions were find by experiments and it is likely that they can be use for every door that is similar.

### 4.2.2 Limits of the method

The method has some limits, first the value in front of the handle are specific to one geometry of the door. If the handle is on the other side as it is showed in the figure 6 with the handle on the left, the positions and orientations need to be find again.
Even if the handle is on the same side, it might be a problem because the positions are hard coded according to the pulling and pushing side of the door. If the door has a different orientation, with reversed pushing and pulling, the door will send back the wrong positions. The pulling side will receive the positions of the pushing side and vice versa.

### 4.2.3 Finding the position for other type of door

To find the position and the orientation for another type of door, some experiments must be done. And the result that are working must be saved in the class door. In order to find the result, the function **print_current_pos** that print the current location of the robot in the door reference system can be called. If a few tests show that a specific position is working, the position and the orientation may be saved.

## 4.3 Passing the door

To pass the door, the robot is not able to use the global planner. The global planner is not working on mobile objects like a door or a chair, and the door is seen as an obstacle that can not be cross in the world model and the global planner can not find a path to go on the other side. To be able to open the door, an other idea that is more dangerous for the robot is used. The idea is to first get the localization of the robot, then compare it to the localization of the destination and then do a movement to go towards the destination. The three steps are in a while loop whose breaking criterion is to bo close enough

to the destination. The destination localization is found using two virtual volumes: **frame_left_point_vv** and **frame_right_point_vv**. Both of their position is get and the mean of the two positions is computed, it represents the middle of the door and then the robot needs to move a little further.

The method is dangerous because the door keep moving even if there is an obstacle.

The same principle is applied in the function created to push the door when it is open.

# 5 Limits of the current work

When performing a high level task with a robot, the script may fail because of physical limits or because of software limits.

## 5.1 Software limits

### 5.1.1 No handling of the failure

The first computing issue is the lack of handling of a potential failure in the current system. Every smach function have an outcome named **fail**, but in most of them, it can never occurs in the code. It is a problem because in reality, every primitive action may fail.

### 5.1.2 Localization problems

The global planner that move the robot in front of the door or the handle relies on the localization of Hero in the world model. However the localization may not be always precise enough for the robot to be at the exact point. The detection of the handle needs to have the RGBD sensor pointed at the handle, and if it is not, the detection is going to fail. The grasping of the handle also requires a precise position and orientation of the robot and if the planner is not precise enough, the grasping may fail. To overcome the two problems, the current location to detect and grasp the handle is based on the distance that is received from the laser data but it is raising other issues other issues linked to the localization: if the robot does not have the good orientation before using the laser data, it may not move towards the handle.

## 5.2 Physics limits

### 5.2.1 Too much force applied

When opening the door , high-intensity forces are applied on some part of the robot, especially on the gripper. When the handle is unlatched, the force applied on the gripper might be too high and the robot may stop for its security. A similar issue may occur if the handle was not unlatched correctly. The robot tries to pull or to push the door, fails and stops. This problem is difficult to overcome by upgrading the software. However other robots with a different hardware may not encounter the issue.

On Hero, there is a currently on the pulling side a slight rotation of the gripper before the unlatching. It has been understood, after some experiments, that one side of the gripper was more sensitive. The rotation allow to apply less force to the sensitive side.

### 5.2.2 Gravity closing the door

If the door is not perfectly equilibrated, it may open without being pushed once it had been unlatched. In that case, the robot has only to unlatch the handle and let the door open by itself. At the opposite, the door may also move back back near the frame after it has being pushed or pulled . The problem is difficult to overcome because a robot like Hero cannot hold the door while it releases the handle. A robot like Spot from the company Boston dynamics is using one of its leg to held the door. [3]

# 6 Future work

The section described ideas that could be use in the future to improve the door opening code.

## 6.1 Use of the force sensor to pull the door

The force sensor on the gripper could be used to pull the door. Instead of extracting an angle between the robot and the door and rotate until the angle is the same as the initial angle like it is explained section 3.2, the function could use the force sensor on the gripper to rotate until a certain treshold of force is applied on the gripper.

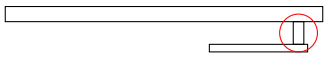


Figure 7: *The red circle is around what is called attachment segment of the handle.*

When the force treshold is reached, it means the gripper is on the attachment segment of the handle and it does not need to rotate anymore. The function that move the robot parallel to the door would also have to evolve.

Writing a new method to pull the door should be an intermediate task because using data from the force sensor is required, however it would improve a lot the function that pulls the door. The success rate of the pulling could be more than 95%.

## 6.2 Use of virtual volumes instead of hard coded coordinates in the door class

Writing the several coordinates of the position the robot should move to in the door class is probably not the best solution. A better idea would be to write the coordinates using a virtual volume in the door model. It would allow to

use the same door class for different type of doors, because all the necessary information would be in the door model.

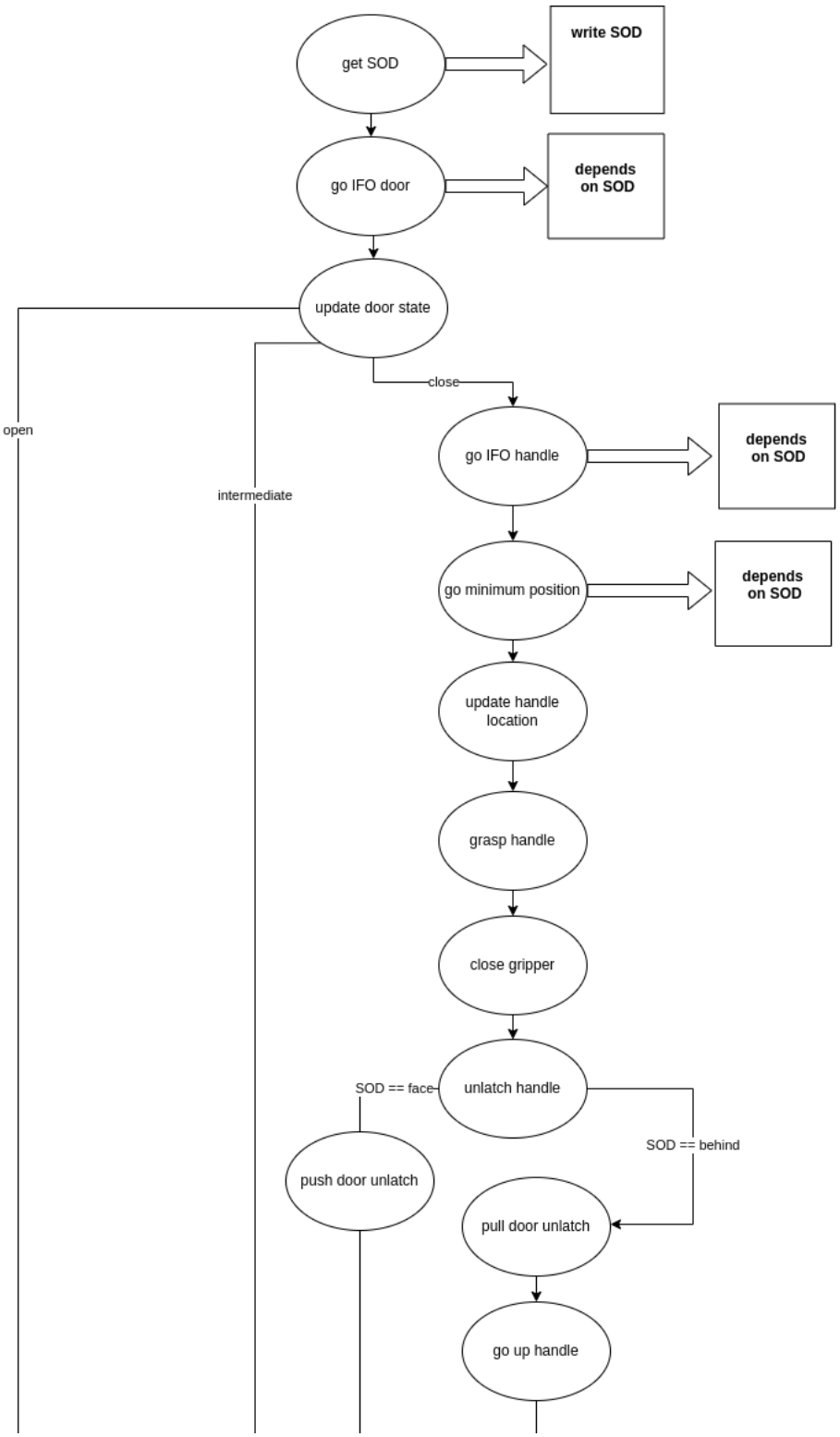
Writing the coordinates in door model and re-write the function in the door class is an easy task. Nevertheless, it is not going to improve the level of the current code, but it is very important to make the code more logic and understandable.

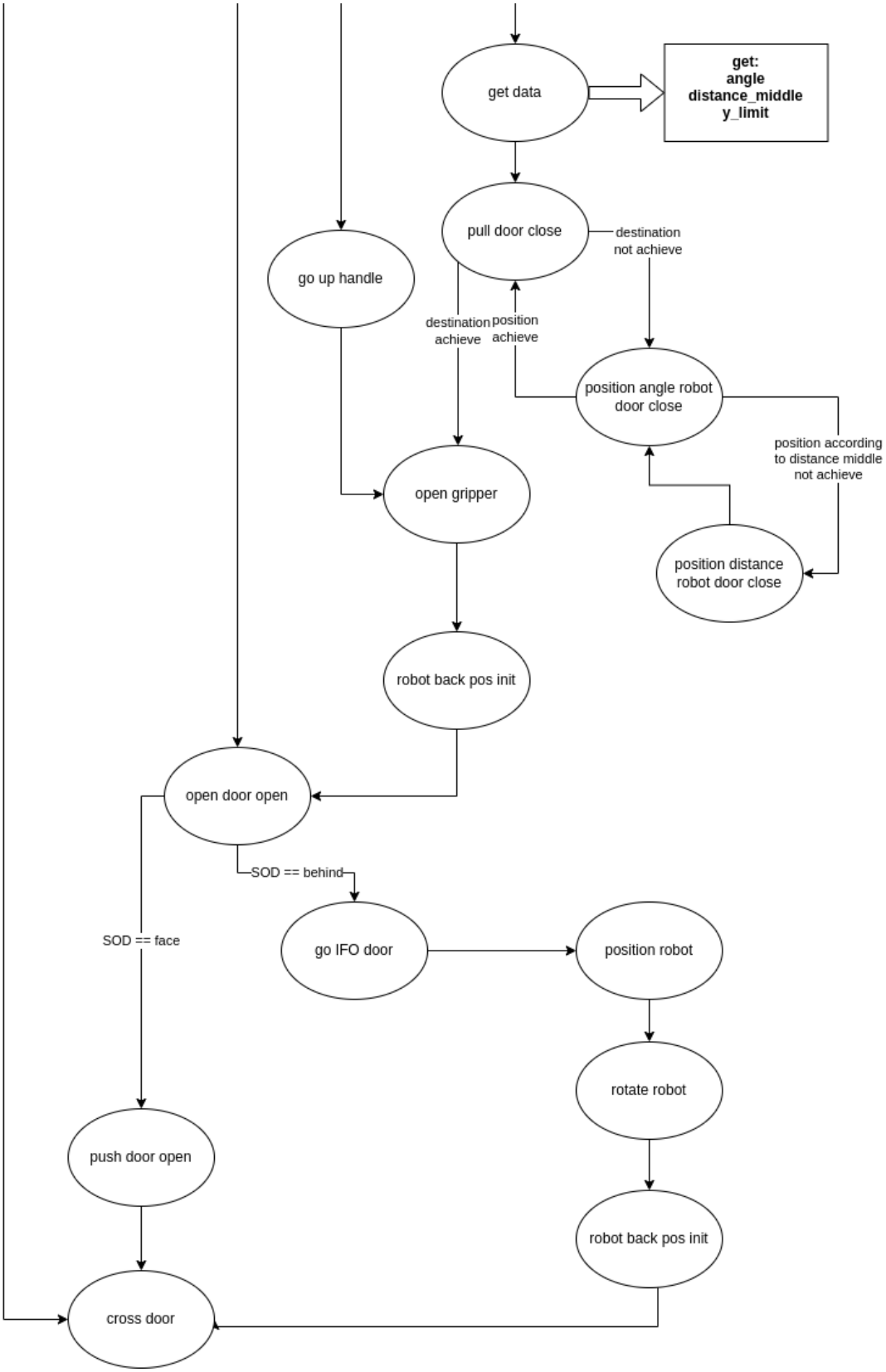
## 6.3 Handling the failure

The non handling of the failure is a problem because each primitive action is likely to fail. Implementing the failure is also important because failing a primitive action does not mean that the door can not be open, it could still be working thanks to another function. For example, if the detection of the handle fails, which can happen because it has already been implemented, the same detection might work after a small rotation of the robot that would allow it to have a better view on the handle. Handling the outcome **fail** is not done by the current implementation. An other example is the global planner, it may not work to go to certain positions, One of the reason could be the presence of obstacle in the world model between the robot and he destination point. It could also be due to the presence of physical object too close to the robot. In the door opening, it happens sometime when the gripper has just been opened after pulling the door and the next step of the robot is to go in front of the door. It can be overlooked by calling a function that move the robot away from the obstacle if the global planner is not working.

Implementing the failure is important to improve the current work because it would make the code more robust. However it is probably a task that is not easy, especially for function where there are no feedback because it is difficult to know if they fail.

# Appendix A: State machine




get data
get:
angle
distance_middle
y_limit
pull door close
destination
not achieve
go up handle
destination
achieve
position
achieve
position angle robot
door close
position according
to distance middle
not achieve
open gripper
position distance
robot door close
robot back pos init
open door open
SOD == behind
SOD == face
go IFO door
position robot
rotate robot
push door open
robot back pos init
cross door

## Appendix B: File example.launch

```
<launch>
  <node name="line_extractor" pkg="laser_line_extraction" type="line_extraction_node">
    <param name="~frequency" value="30.0" />
    <param name="~frame_id" value="hero/base_link" />
    <param name="~scan_topic" value="/hero/base_laser/scan" />
    <param name="~publish_markers" value="true" />
    <param name="~bearing_std_dev" value="1e-5" />
    <param name="~range_std_dev" value="0.012" />
    <param name="~least_sq_angle_thresh" value="0.0001" />
    <param name="~least_sq_radius_thresh" value="0.0001" />
    <param name="~max_line_gap" value="0.5" />
    <param name="~min_line_length" value="0.7" />
    <param name="~min_range" value="0.25" />
    <param name="~max_range" value="2.5" />
    <param name="~min_split_dist" value="0.04" />
    <param name="~outlier_dist" value="0.06" />
    <param name="~min_line_points" value="30" />
  </node>
</launch>
```

# References

[1] R. Janssen, E. van Meijl, D. Di Marco, R. van de Molengraft, and M. Steinbuch. Integrating planning and execution for ros enabled service robots using hierarchical action representations. *In 2013 16th International Conference on Advanced Robotics (ICAR)*, pages 1–7. IEEE, 2013.

[2] Wikipedia, High and low level,
url: `https://en.wikipedia.org/wiki/High-_and_low-level`

[3] YouTube, The Guardian: New dog-like robot from Boston Dynamics can open doors, url: `https://www.youtube.com/watch?v=wXxrmussq4E`