4SC020 Mobile Robot Control

Design Document

Quartile 4 Year 2020-2021

B.H.T. Bindels 1246348
S.B.M. van den Broek 1252011
T.S. Ickenroth 1232296
L.T.M. Verstegen 1252488
L.M.J. van Dooren 1249169

Course coordinator
Dr. ir. M.J.G. van de Molengraft

Tutor
Ir. H.L. Chen

Eindhoven, May 4, 2021

Requirements

The requirements are defined as aspects what the system *should* do. The assignment consist of two tasks, which are the Escape Room and Hospital challenge. The design document focuses mainly on the Escape Room. In the list below, several requirements are listed which the PICO should cover. On top of that, additional requirements are added to the list in order to complete the Hospital challenge.

- The robot should cross the finish line as fast as possible.
- The robot is able to operate autonomously.
- PICO must not touch the wall, the minimum distance between the wall and the robot should not be smaller than 100 mm.
- The system should not remain in idle position for more than 30 seconds.
- The task must be completed within 5 minutes, having two attempts.
- The robot must drive in the direction of the corridor once it detects a hole in the wall $\longrightarrow$ once it detects nothing in front AND left/right it must turn left/right corresponding to which direction is empty.
- In case a collision does occur, the robot must drive in the direction opposite to the side of the robot where the collision occurred.

Additional requirements for the hospital challenge;

- The robot must distinguish between static objects and dynamic objects and the way in which it avoids the object.
- The robot can find target cabinets in order to deliver packages.

In the Figure 1, a requirements tree is visualized where the connections between them are identified.

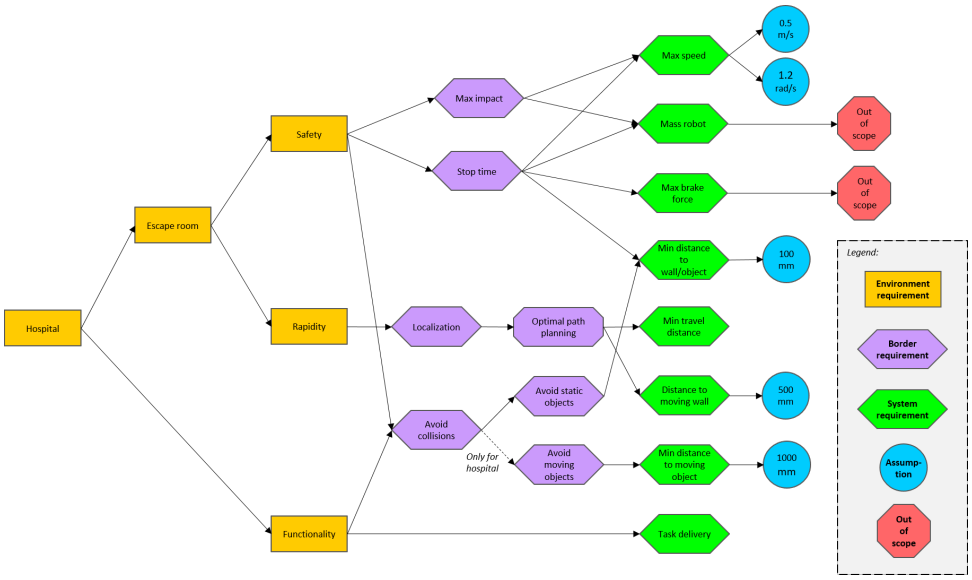

Figure 1: Schematic overview of the requirements and purposes in a tree diagram.

Components

The PICO robot that is used is a telepresence robot from Aldebaran it consists of the following hardware:

- Laser Range Finder (LRF)
- Wheel encoders (odometry)
- Holonomic base (omni-wheels)
- Intel i7
- Running Ubuntu 16.04 (simulator runs on Ubuntu 18.04)

Specifications

System

- Maximum speed of PICO is: 0.5 $[m/s]$ in translational direction, and 1.2 $[rad/s]$ in rotational direction.
- The robot has a holonomic base, meaning that it can move forwards, backwards, sideways, and rotate around its own axis.
- LRF has a field of view of 4 radians and is capable of measuring distances reaching from 0.1 to 10 meters. With this information the robot is capable of detecting and avoiding objects and walls.
- Wheel encoders are used for odometry data, location of the robot with respect to the initial position can be retrieved from this data.

Environment

- Shape of the environment is rectangular, exact dimensions will not be given.
- PICO starts at a random position in the room, not necessarily facing the exit, nor a straight wall.
- Orientation of the corridor will be perpendicular to the wall.
- The wall of the corridor will be open on the far end.
- The walls might not be perfectly straight, the corners might not be perfectly perpendicular, and the walls of the corridor might not be perfectly parallel.
- The width of the corridor will be somewhere between 0.5 $[m]$ and 1.5 $[m]$.
- At the exit, the finish line will be located more than 3 meters into the corridor. The walls of the corridor will be a bit longer.

Software

- Software can be updated with one easy command, e.g. '*git pull*'.
- Software can be compiled using '*cmake*' and '*make*'.
- To start the software only one executable has to be called.
- Software will be updated on the robot before the challenge starts.

Functions

This section provides a visualization of the Finite State Machine (FSM) together with a description of all states included in the escape room challenge. Please note that the FSM of the hospital challenge will have a different setup

- **Localizing**: This is the initial state. The robot tries to find the corridor by rotating max 90 degrees to identify all walls of the escape room. If it succeeds, state **Aim2corridor** is entered. If it does not find the corridor, state **go2wall** is entered.
- **Aim2corridor**: Here the robot rotates such that it faces towards the entrance of the corridor. When its aligned, state **Travel2corridor** is entered.
- **Travel2corridor**: The robot starts travelling towards the entrance of the corridor by moving in x and y-directions. When a wall is closer than 300 mm to the robot, it moves back to the **Localizing** state to re-identify to location of the corridor. When everything goes right, the robot will find its way to the entrance of the corridor after which state **Align_corridor** is entered.
- **Align_corridor**: Here the robot is positioning itself such that it can drive in the corridor by a straight line. When its positioned parallel to the corridor walls, state **Travel_corridor** is entered.
- **Travel_corridor**: The robot drives towards the finish line keeping track of the distance between itself and the walls. When a wall comes closer than 200 mm to the robot it will stop and re-align itself with the walls in state **Align_corridor**. When the rear wheels have crossed the finish line the final state is entered. Here the robot will simply stop and celebrate its victory.
- **go2wall**: Here, the robot start driving towards the nearest wall. When the distance between the wall and the robot is less than 500 mm, the next state becomes active.
- **Align_wall**: The robot stops, so that is does not bump into the wall, and starts rotating until the base of the robot is parallel with the nearest wall such that the robot can drive in forward direction parallel to the wall.

- **Move**: The robot starts driving along the wall trying to keep the same distance perpendicular to the wall. If for some reason the distance between robot and wall becomes smaller than 200 mm, the **Reposition** state becomes active. While driving, the robot scans the environment and checks if it can find the corridor. When the corridor is identified, the state **Aim2corridor** becomes active again. If the robot comes close to a corner (within 500 mm) the state **Corner** becomes active.
- **Reposition**: The robot will reposition itself parallel to the wall with a distance of 500 mm and switch then back to state **Move**.
- **Corner**: The robot stops, rotates 90 degrees, and will then align itself with the wall in state **Align_wall**.

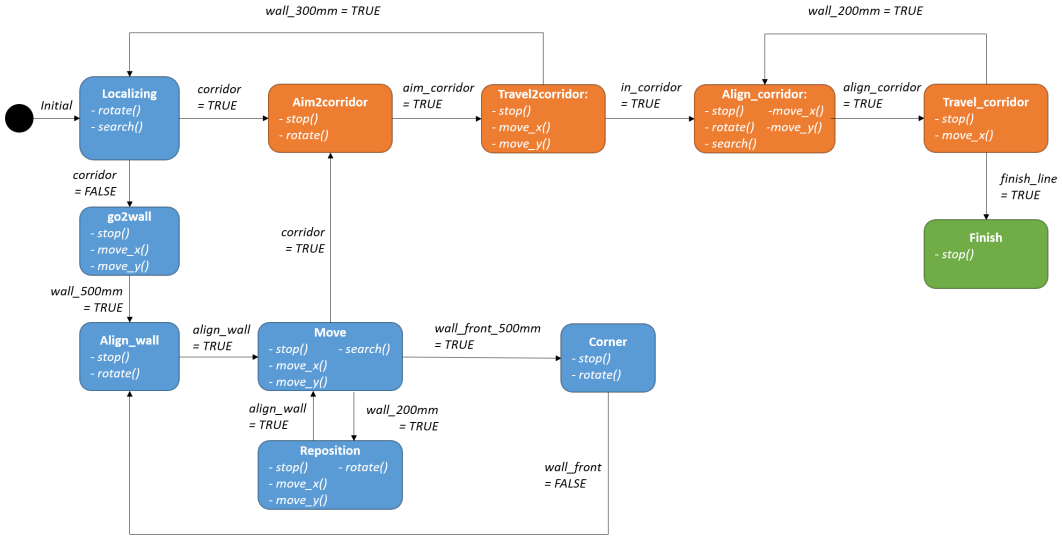

Figure 2: Schematic overview of the Finite State Machine displaying the states and the booleans.

Interface

PICO receives information of the real world by sensing the world with its sensors. In Figure 3, the real world is indicated with the block 'world'. The finite state machine only works properly if it receives and sends out the correct information from/to PICO. This is taken care of by the interface. The reconstruction part of the interface ensures that sensor data is translated into data which can be used by the finite state machine, and the actuation part of the interface ensures that decisions taken by the finite state machine translate into function calls understandable by PICO. This is visualized in Figure 3.

The main functions in the interface will be:

- From the LRF data determine where the walls are and what the smallest distance to each wall is.
  - In case a gap is sensed between two parallel oriented walls, e.g. both vertical, PICO should know that the gap corresponds to the corridor and therefore the corridor bool is set to TRUE. Then PICO should rotate such that he is facing towards the location of the corridor, after which the aim_corridor bool is set to TRUE.
  - From the orientation of the walls and the current orientation of PICO he should be able to determine how he is oriented with respect to the walls and correspondingly make the align_wall bool TRUE when he is aligned to the wall that he is closest to.
- Keep track of the traveled path, i.e. monitor the wheel encoder data such that PICO can remember where he has already driven.
  - When PICO has entered the corridor, he should orientate himself parallel to the walls of the corridor and monitor the distance traveled such that he drives parallel to the walls for at least 3 m in order to reach the finish line.
- The actions which are specified in the finite state machine should be translated into function calls which can be directly used by PICO.

– If PICO needs to rotate, the two main wheels should turn in opposite direction.
– If PICO needs to move in x or y direction, either the main wheels or sub wheels need to move. If PICO needs to move diagonally, so both in x and y direction, the main wheels and sub wheels need to move simultaneously.
– If PICO needs to stop, all wheels should gradually slow down until zero velocity.

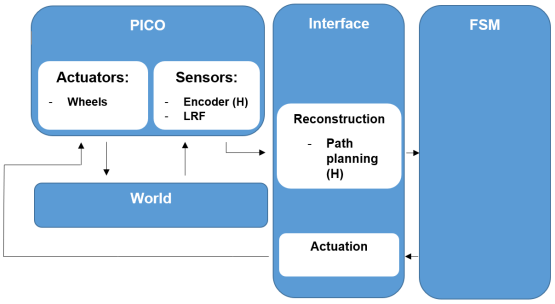

Figure 3: Interface scheme