# MRC Design Document

C. Conrad (1016008)
E. Szücs (1536419)
L.S. Hagendoorn (1013526)
M.H.B. Teurlings (1504274)
S.J. Leemrijse (1009082)
W.M. Klip (1015501)

May 2021

# 1 Requirements

In Figure 1 the environmental, border and system requirements are represented in yellow, blue and green respectively. The red enclosed part is specific to the hospital challenge. Three stakeholders are considered: the hospital personnel, the patients and the software engineers.

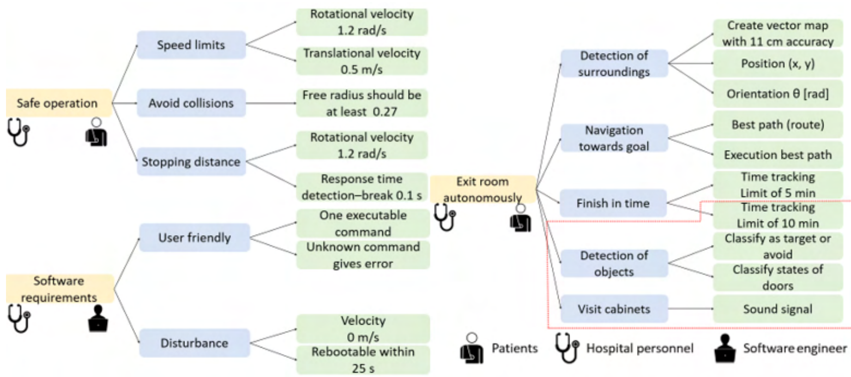


Figure 1: Environmental, border and system requirements with their corresponding stakeholders. Red enclosed part specific to the hospital challenge.

*Safe operation:* PICO operates in an environment where humans are omnipresent, therefore it is required that it operates in a safe manner. This means that PICO obeys certain translational (0.5 m/s) and rotational (1.2 rad/s) speed limits that are considerably less than human walking speed. Also, collisions with static and dynamic objects should be avoided. Since PICO is 0.41 m in width by 0.35 m in length, the free radius should be at least $\frac{1}{2}\sqrt{0.41^2 + 0.35^2} = 0.27$ m. Moreover, the maximum stopping distance should be at least 0.32 m, which is the free radius plus an allowed reaction time of 0.1 s dependent on the processing speed of the robot hardware at 0.5 m/s. *Stakeholders:* hospital personnel, patients.

*Software requirements:* The software should be user friendly, which is accomplished by the usage of one executable command and provision of error messages for unknown commands. In case of a disturbance the robot should stop moving and reboot itself within 25 s, in order to not stand still for longer than 30 s. *Stakeholders:* hospital personnel, software engineers.

*Escape room:* PICO must exit the room within 5 minutes, from an unknown starting position. It should create a vector map for rejective field mapping using the laser scanner and odometry data with accuracy of at least $d = \sqrt{2r^2} - r = 0.11$m to turn in corners, as visualised in the appendix. From this data position and orientation are also determined. The map, together with position and orientation will allow PICO to traverse the room. From the generated map it should configure the shortest, safe path and subsequently PICO should execute this route. *Stakeholder:* Software engineers.

*Hospital challenge:* The main objective is to visit all cabinets in the specified order, within a time limit of 10 minutes. This order should be given to PICO through one command. In this challenge, PICO should detect dynamic and static objects and classify those as either target or avoid. The state of the doors (open or closed) should be detectable and acted upon by the robot in order to prevent damage. Furthermore, PICO should visit and face the cabinets, whereafter it should indicate its success with a sound signal. *Stakeholders:* hospital personnel, patients.

## 2 Components and Specifications

We decided to describe both *specifications* and *components* in the same section since they are strongly linked together. In this year's version of the course, we are going to use a simulation which has integrated only the key features of the PICO robot. This means that we are going to use only the following components that give certain specifications:

*Laser Scanner:* It is capable of detecting anything that reflects light in a radial range of 0.01-10m and an angular range of 270° ($\frac{3\pi}{2}$rad). Based on the software that we are going to implement, with the laser data we can detect walls, edges, corners, flat spaces, hallways etc, but also moving object as an opening door or a passing human, which will be a key ingredient in completing the upcoming challenges.

*Odometry sensor:* Returns x, y position and orientation angle $\theta$ relative to the robot's initial situation based on the wheel encoders, thus we can also obtain the speed and the path that it took from its initial position.

*Holonomic wheels:* Moves the robot in any direction in the 2D space of the environment in which it operates.

*Environmental components:* In its environment, the robot should avoid objects that can be static, such as walls, boxes, closed doors, or dynamic, such as humans walking around.

## 3 Functions and Interfaces

We have chosen to categorise our functions based on their purpose, and they are as follows:

*Data acquisition:* In order to make use of the other functions, we have to start by measuring the environment in which the robot operates, thus the *data acquisition* function.

*Object detection:* The robot has to be able to recognise static, as well as dynamic objects in its path. In order to do so, it has to make use of the laser scanner which is, in this simulator, the only component capable of object detection. A way to do this is by creating patterns, such as constant data values would indicate the presence of a wall, while for the dynamic objects, such as a passing human, there would be an increase of the detected values if the object is approaching and a decrease if the object is moving away from the sensor, while compensating for its own speed and direction of course.

*Communication between modules:* All the data is saved in the Data Bank module. To write, or retrieve data from the Data Bank, "getters" and "setters" type of functions are needed.

*Robot actuation:* The robot has to be actuated towards its goal, so either exiting a room or going to a certain place in its environment. The obstacles that are coming in its way should be avoided based on the laser scanner, while bearing "in mind" the final goal where it should arrive.

*Updating state:* Following a series of inputs from the laser scanner and odometry sensor, the finite-state machine updates the state of the robot and the environment in real-time.

An overview of how these functions are distributed among the different software modules is shown in Figure 2.

It is important to have a clear structure when writing software to execute a complex task, especially when the code is written by multiple contributors. The main focus, as illustrated in Figure 2, are the data flows between modules. The basic concept is to have one main data bank which has all relevant information available such that each function can independently access the data it needs. The action planner can then simply call any function with minimal input parameters.

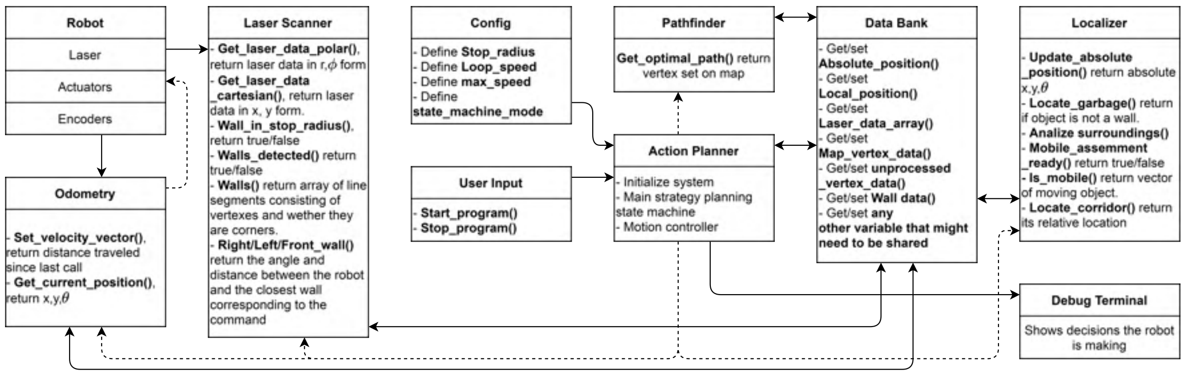


Figure 2: Proposed data flow diagram of the system, dotted lines represent commands and solid lines represent (large) data flows, all data is first collected by the data bank before it is past on

Any new data that are generated by function calls will get updated immediately in the data bank to ensure that there is never mismatching or outdated data. Apart from this basic structure the action planner sets certain configuration parameters at the start of the code as defined in the `.config` file to easily setup tests. A proper debug window/terminal will help find any errors in the decision making process of the robot. Lastly, the user input block will allow the robot to start/stop its autonomous cycle. Note that not all functions, as named in the interface overview of Figure 2 are required for the escape room competition, some thought has already gone into the final competition as well.

*Behaviour:* The action planner is essentially a state machine. Since the escape room competition doesn't require complete positional awareness of the robot and project time is limited, a simple path planning algorithm will be implemented instead of, for example, a SLAM algorithm. The simple preliminary state machine shown in Figure 3 gives a rough idea of the decision logic. It should always find the corridor of a room and exit through it, whilst making very little assumptions about the shape of the room. Notice that some advanced function are still required though, like the corridor detected function. These function will of course be reused in the final challenge.

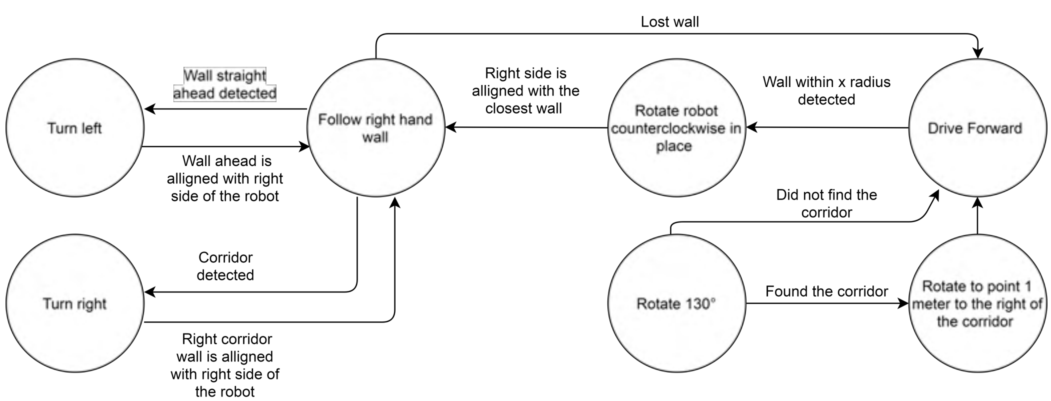


Figure 3: State machine structure

# 4 Appendix

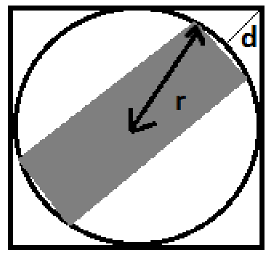


Figure 4: Accuracy $d$ needed for the odometry data obtained by the robot (grey).