Eindhoven University of Technology

Course 4SC020

Embedded Motion Control

Group 8:

Initial Design

S. Bouwmans (0892672)

S.A.M. Rangan (1279785)

J. Baeten (0767539)

Y. Le grand (1221543)

M. Heijnemans (0714775)

Tutors

M.J.G. v.d. Molengraft & H.J.P. Bruyninckx

May, 2018

Contents

1 Introduction

Robots which will fullfill specific tasks without human interference are getting more and more attention every day. Think of an sorting machine, an autonomous car which drives without a driver and even a whole robotics soccer team. However, these robots are not able to anticipate rationally on unexpected situations. Every movement of the robot is designed and programmed beforehand by humans, which is called embedded motion control. In this course with the same name, a robot has to be programmed in a way that it will fullfill a specific task as indicated in section 2. Before the real programming can start, an initial design has to be made which consist of a list about the requirement, specifications, functions and so on the robot has to have which will be given in this report. This document will therefore exist as guideline for the programming of the robot, however, because it is an initial design, the final design is expected to differ from its initial design.

2 Goal

In this course, the PICO robot, which uses the UBUNTU 16.04 opperating system and the program langueges C++, as shown in figure 6 has to be programmed to finally fullfill the so called hospital challenge, indicated as HC. However, as intermediate step, the escaperoom challenge needs to be completed which is indicated as EC. A brief description of the two challenges is given below. As could be seen, most parts of the escaperoom challenge are also present in the hospital challenge, making it a good intermediate control step of the movement of the robot for the hospital challenge.

Figure 1: The PICO robot which has to be programmed to fullfill the escaperoom challenge and the hospital challenge

2.1 The escaperoom challenge

In the Escaperoom challenge, PICO has to escape a rectangular room from a random position and orrientation. This room has one opening that leads to a corridor with a width between 0.5 and 1.5 meters. After more than 3 meters, there is a finish line in the corridor. PICO has to pass this finish line within 5 minutes, but as fast as possible, without bumping into the wall or standing still for 30 seconds. Furthermore, PICO is not allowed to move faster than 0.5 m/s translational and 1.2 rad/s rotational. An example of a possible room is given in figure 2.

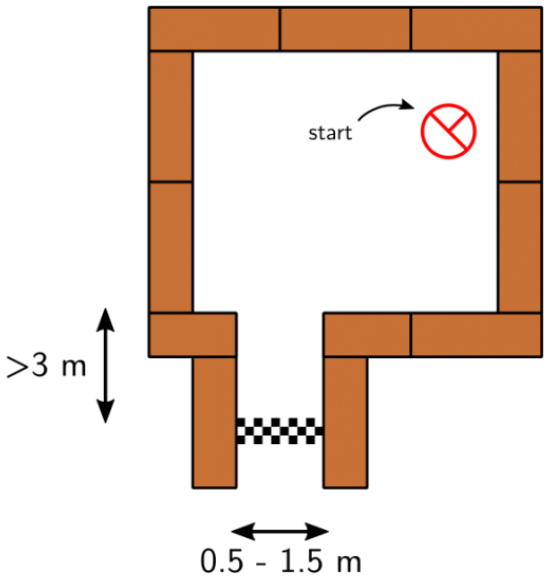

Figure 2: A possible scheme of the Escaperoom challenge

2.2 The Hospital Challenge

In the Hospital challenge, PICO will start in the so called hallway. Besides the hallway, there are 3 to 6 rectangular rooms PICO can explore. A possible setup of the environment of this challenge is given in figure 3. Now, instead of going to a specific finish line like the Escaperoom challenge, PICO is allowed to move through the environment to make a map before parking backwards behind the wall in its starting position. Than, a object will be placed within a room, which position will be already specified beforehand with a hint. PICO than has to find the object and must stop moving close to the object to complete the challenge. This all has to happen as fast as possible and under the same restrictions of movement and time as in the escaperoom challenge.

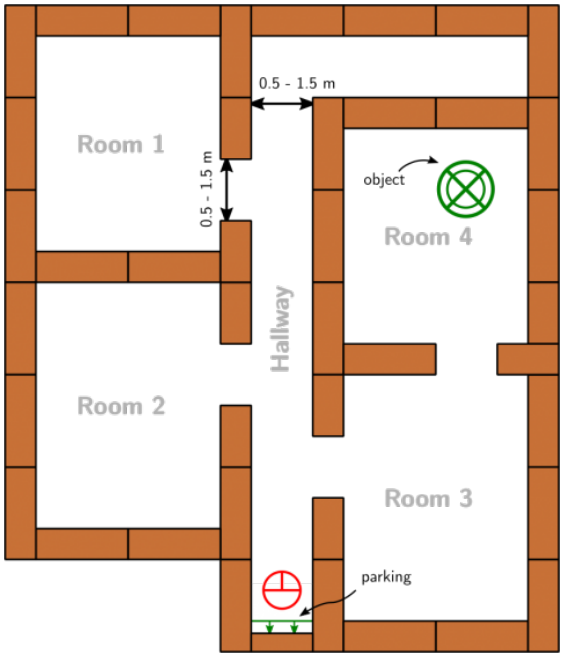

Figure 3: A possible scheme of the Hospital challenge

3 The Initial Design

For the Initial design, we decided to first mainly focus on the Escaperoom challenge. This is done because the movement of PICO in the Escaperoom challenge is at the basics of the movement in the Hospital challenge. If the basic movements of PICO are programmed and PICO can move fast through a room without colliding with the walls, the add ons needed for the Hospital Challenge like the mapping, can be easily added later to the programming code. However, some aspects we have already thought of that are specifically for the hospital challenge and not for the escaperoom challange are already listed and indicated with HC.

3.1 Requirements

- No ”hard” collisions with the walls
- Find and exit the door of a room
- Fullfill the task within 5 minutes
- Do not stand still for more than 30 seconds
- Fullfill the task autonomously
- Map the complete environment (HC)
- Reverse into wall position behind the starting point after an exploration of the environment (HC)

- Find a newly placed object in an already explored environment (HC)

3.2 Nice to haves

- High speed of the movements
- Efficient way of the movements
- No collisions with the wall
- Interoperate the provided hint of the objects position (HC)

3.3 Functions

- Driving
  - Translating
  - Rotating
- Comparing Data
  - For trajectory planning
  - For newly placed object recognition (HC)
- Line fitting (least squares in C++)
- Trajectory and action planning
- Configuration
- Coordination
- Communication
- Mapping (HC)
- Parking (HC)
- Sleep (HC)
- Mediation (HC)

3.4 Components

- Sensors
  - Laser Range Finder (LRF)
  - Wheel encoders (odometry)
- Actuator

  - Holonomic vase (omni-wheels)
- Computer
  - Intel i7
  - Ubuntu 16.04 with ROS Kinetic
  - C++ programming language
- programming specifications
  - Qt creator as IDE
  - Git as version control
  - PICO simulator as simulator
  - Group wiki as documentation
- Software (various executables that run during the lifetime of the components
  - Worldmodel
  - Detection
  - Drive control
  - Main executable

3.5 Specifications

- Maximum translational speed of 0.5 m/s
- Maximum rotational speed of 1.2 rad/s
- Translational distance range of 0.01 to 10 meters
- Oreintation angle range of 02 to 2 radians or 229 degrees approximatly
- Angular resolution of 0.004004 radians
- Scan time of 33 milliseconds

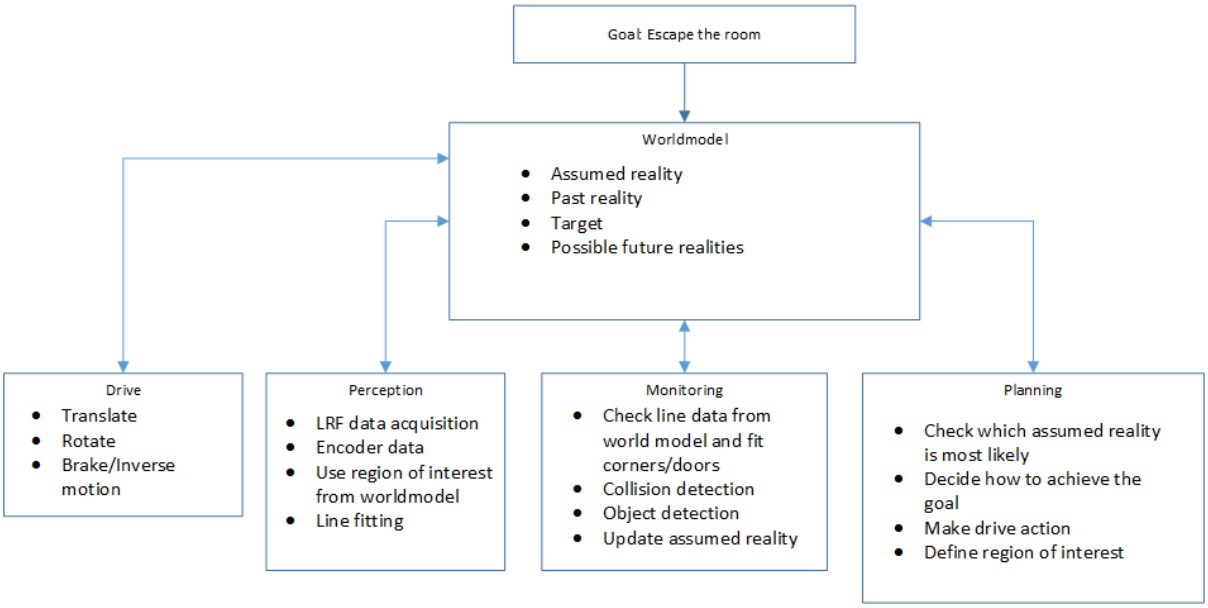

Figure 4: The Task Skill Motion scheme of PICO for the Escaperoom challenge

3.6 Interfaces

The task skill motion framework is given in figure 4. A description of this is given below

- World Model
  - Initialize
    - Initial position
    - Rectangular Room assumption
    - Inside room for EC, inside hallway for HC
  - "Height" map
  - Maximum speeds/turns
  - Position
    - Coordinates
    - Speed
  - Corridor (0.5 to 1.5 meters)
  - Exits (finish line is more than 3 meters inside the corridor) (EC)
  - Modelling
    - Wall (might not be perfectly straight)
    - Corner (might not be perfectly perpendicular
    - Door

        * Room (HC)
        * Object (HC)
        * Hallway (HC)

- Perception
    - Sampling 10-100 Hz
    - 170 degrees normally and 240 degrees for exceptional cases like passing through doors and parking
    - Odometry/encoder for parking

- Plan/Control
    - States
        * Forward full/speed
    - Events
        * Wall
    - Trajectory
        * Selecting best trajectory for certain speed

- Monitoring (Hypotheses)
    - Locating wall and configure control
    - Looking for further geometry and act as per plan
    - Discarding Non-ROI data
    - Neglecting all measurements according to update frequency

More on this is given in figure 5

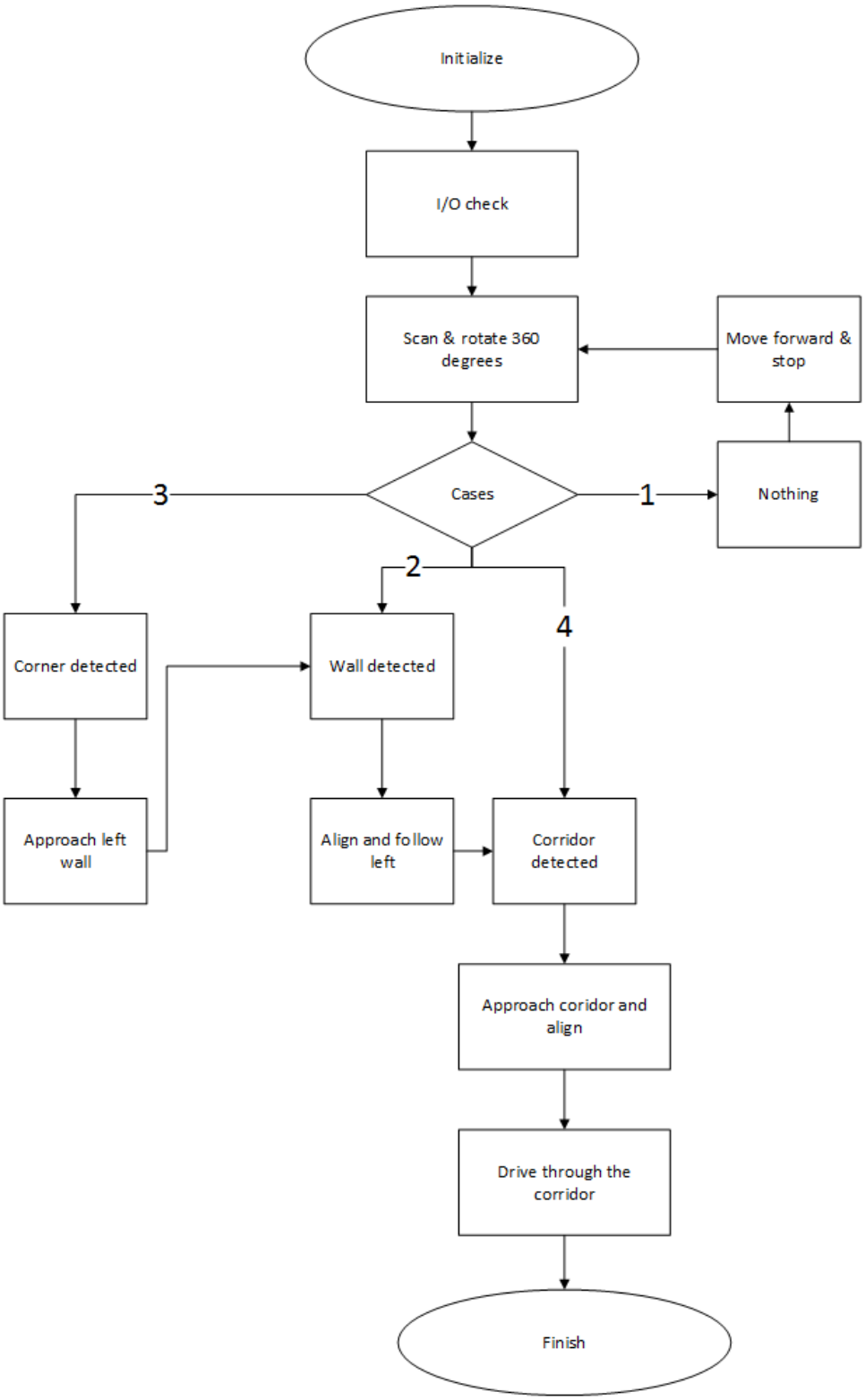

Figure 5: A draft of the possible events and control actions of PICO

4 Data Structure

4.1 Perception

input:

```
struct LaserData
{
    double range_min;
    double range_max;

    double angle_min;
    double angle_max;
    double angle_increment;

    std::vector<float> ranges;

    double timestamp;
};
```

From this date we have the information on the relative position of the `n_beams` data points relative to PICO. This includes the distance to PICO enclosed in:

```
std::vector<float> ranges;
```

and the angles between the beams:

```
double angle_increment;
```

The procedure is to transform the data cluster using the following functions:

1. TransformCartesian
   This transform the ranges to x and y columns coordinates by multiplying them with `angle_increment`. This outputs an array of several 2d arrays corresponding to the x and y coordinates.

2. SortPoints
   Sorts the points using the 'Split and Merge' algorithm into groups. Further segments groups iteratively by taking a line between first and last points of groups and measuring perpendicular distance of each point to this line.

3. LSE
   use least squares to define the lines through the separate data clusters. This outputs an array of vertices ordered according to the corresponding coordinates of TransformCartesian.

4. IdentifyNode

Make begin and end node by choosing points on the lines that are closest to your end data points. The output is n points which are further described below.

Output:

ArrayNodes:

A 3xn array describing the x and y coordinates and whether the node is a corner or an end point. Corners are points where two line segments meet. End points are open ending of line segments.

$$\begin{array}{c} \\ n_0 \\ n_1 \\ n_2 \\ ... \\ n_n \end{array} \begin{array}{c} \begin{array}{ccc} x & y & c \end{array} \\ \begin{pmatrix} x_0 & y_0 & 0 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ ... & ... & ... \\ x_n & y_n & ... \end{pmatrix} \end{array}$$

In the third column 0 indicates if it is an end point and 1 indicates a corner.

ArrayLineP:

This is a 2 column array that indicates lines. Each row consists of two nodes that are on the same vertex. An example is given below:

$$\begin{array}{c} \\ l_0 \\ l_1 \\ l_2 \\ ... \\ l_? \end{array} \begin{array}{c} \begin{array}{cc} n_a & n_b \end{array} \\ \begin{pmatrix} n_0 & n_1 \\ n_1 & n_2 \\ n_3 & n_0 \\ ... & ... \\ n_w & n_z \end{pmatrix} \end{array}$$

The column length is not fixed.

4.2 Monitoring

The monitoring stage of the software architecture is used to check if it is possible to fit objects to the data of the perception block. For instance by combining four corners into a room. The room is then marked as an object and stored in the memory of the robot. The same is done for doors and corridors. This way it is easier for the robot to return to a certain room instead of exploring the whole hospital again. For the escaperoom only the door has to be detected in the data. Also the monotoring skill will send out triggers for the planning block if the robot is probably colliding with an object/wall. Input:

- ArrayNodes
- ArrayLineP

Functions:

- Detect if and how far a wall is in front of the robot
- Detect object and distance
- Make objects of rooms the rooms with coordinates and dimensions
- Make objects of the doors with their coordinates and dimensions
- Make object of the hallway

Output:

- List with objects and their coordinates relative to the starting position.
- Trigger for planning if robot is possibly colliding with an object
- Trigger for planning if robot is driving towards a wall while following one

4.3 Planning

For the driving a few datastructures are needed. A speed vector (in x and y relative to Pico) will be translated to forward, sidewards and rotational speeds for Pico. To get this speedvector the following flow of data is shown in the table below. For the function/ algorithms there are several parameters with can be defined as constants. For example the range in which a potential field force exists. These constants can be defined inside a object which contains all constants. Or, alternatively, it can be stored in typedefs.

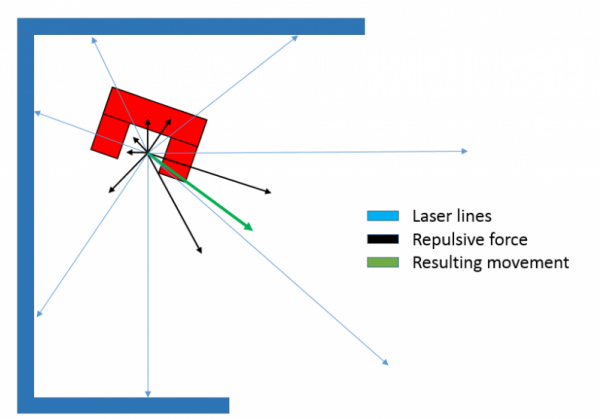

Figure 6: Visualization of potential field repulsion, source: EMC 2017 Group 10

data in	function	data out
worldmodel.map worldmodel.state	find middle door	setpoint
worldmodel.error worldmodel.state	check error $<$ error_acceptable find least explored direction	setpoint
worldmodel.setpoint	potential-field repulsion	speed vector

In this table the state is an integer. 0 is intitialising, 1 is exploring, 2 is going through the exit. The setpoint contains a two arrays of floats: one for the x and y coordinate relative to the robot and another array with the direction in which Pico should be heading in this setpoint.