Eindhoven
University of Technology

Eindhoven

Embedded motion control

4SC020

Design report

Author:
R. Beenders
M.P. Bos
S.J.M. Koopmans
M. Cornelis

ID-number:
0717576
0774887
0746766
0772526

April 26, 2016

Design

This report describes in a concise manner the steps that were taken to come to the first draft of the program-architecture for the PICO robot. Firstly the requirements are stated on which the program-architecture is based. Followed by a global explanation of the structure of the eventual program. The components of the robot are shortly discussed followed by the (expected) specifications. Finally the interface is discussed.

Requirements

- The robot has to operate autonomously
- The robot is not allowed to collide with anything
- The robot must find the door and request to pass it
- The robot has to solve the maze within 7 minutes and within 2 tries

Functions

The global structure of the program-architecture can be seen in Figure 0.0.1. As becomes clear from the picture the code was created from the top down. The requirements were translated into a clear goal. First it was established what was needed to achieve the goal. To determine the location of the door and to solve the maze some kind of model of the world is needed. To create this model and move through the world tasks are needed. The door is found and the maze solved through a combination of the world model and tasks. To perform the tasks several skills are required. What skills are available is limited by what the robot can do. Several skills are defined based on the sensors and actuators present in the robot. A more in-depth explanation of what the tasks and skills mean can be found on the next page.

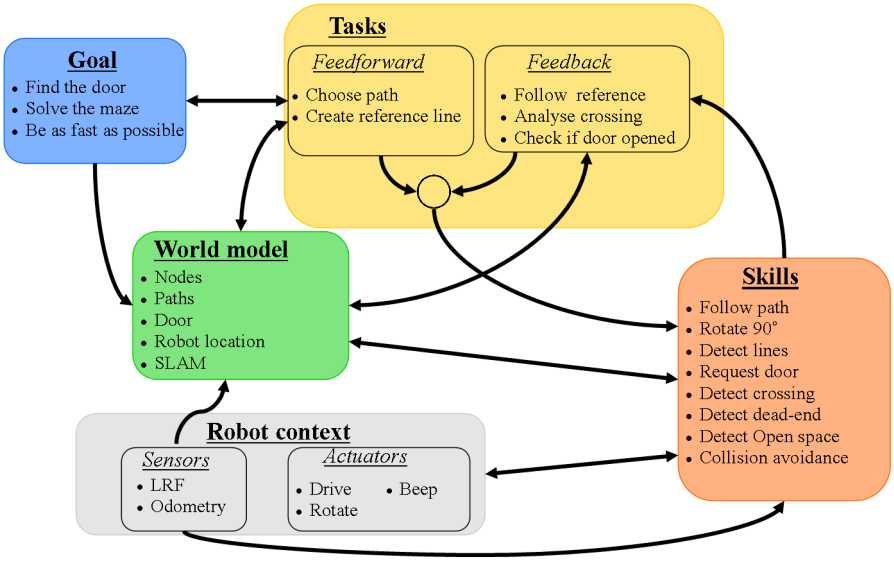

Figure 0.0.1: *Program-architecture*

Goal

The requirements that directly describe the goal are encompassed below:

- The robot must first find the door and request to pass it
- The robot has to solve the maze as fast as possible

The collision avoidance is incorporated on a lower level because it has to hold for all tasks, this means collision detection and avoidance will be designed as a skill that is used for all tasks involving movement. The requirement that the robot has to operate autonomously is incorporated per default.

Environment context

Contains an array of structs which describe the location, amount of connections and state of orientation-points. Also contains the location and rotation information of the robot. Whether locations are described relative to each other or to some absolute origin is to be decided and will depend on the approach that is used to move the robot through the maze and update the world-map. Experiments will show whether the odometry can be sufficiently corrected by using the laser or some other form of calibration in orientation-points can be applied.

Tasks

The tasks can be divided into two groups: feedforward and feedback. Feedforward tasks are 'choose path' and 'create reference'. 'Choose path' will let the robot decide which way to drive from an intersection. 'Create reference' will create a straight line (between the walls) which the robot should follow. Feedback tasks are 'analyse crossing', 'follow reference' and 'check if door opened'. 'Analyse crossing' will allow the robot to check which directions are open and which are not. The world has four defined directions: North, West, South, East (N,W,S,E). The robot should also determine if the intersection is actually a dead-end or open-space. 'Follow reference' should let the robot follow the created reference line by use of a controller. 'Check if door opened' should make the robot wait for five seconds to see if the door actually opened (since the robot can't always see the difference between a door and a wall).

Skills

The robot has several skills which are driven by the robot (actuators and sensor), which are used in tasks. The following six skills, defined as: 'follow path', 'rotate 90 degrees', 'detect lines', 'request door', 'detect crossing/open-space/dead-end' and 'collision avoidance'. Follow path allows the robot to drive forward along a straight path towards the next node. Rotate 90 degrees makes the robot rotate for (about) 90 degrees, so that it will face the next direction (N,W,S,E). Detect lines should create straight lines out of the raw data received from the LRF. This will allow the robot to create a local version of the world with straight walls. Request door uses the beep sound which will make the door open. There are some requirements for the door to open (for example, the robot has to stand still). Detect crossing/open-space/dead-end should let the robot see when it is done travelling along a path, and it has found a crossing (or open-space or dead-end). This can be done by noticing that the walls on the left and right have disappeared. Collision avoidance should be a sort of emergency brake, which makes the robot abrupty stop once it gets too close to a wall.

Robot

The robot has some actuators and sensors, which will be described in the next chapter.

Components

The robot consists of both actuators and sensors. The actuators are the wheels, which can drive and rotate the robot. Besides that there is a beep function which can be used to open doors. The sensors are the Laser-Range-Finder (LRF) which can detect the distance to nearby walls, and the odometry which can measure how much the wheels rotate.

Specifications

The robots maximum speed is 0.5 m/s translational and 1.2 rad/s rotational. The LRF has a width of about 4 rad (from -2 to 2 rad), with a resolution of about 1000 points. The actual measurement accuracy of the odometry is not very important, since the wheels slip a lot. The two driving wheels aren't parallel to each other, which causes this slip. The accuracy of the information found by this odometry could therefor be very low.

Interfaces

When the program is running it should print which task the robot is performing. When a task has finished correctly a complete message is displayed. Any errors that occur will be printed with explanation of the error. A timer is used to determine how long tasks take. This information can be used to find what situations make tasks take longer and which tasks take longer in general. The code can then be optimized accordingly.
Besides text there will also be a visualisation of the world-mapping. The detected lines will be drawn until a crossing is analyzed and will then be replaced by a node. As the robot solves the maze, the nodes will be connected and a node-map of the maze will be slowly built.