TU/e Mechanical Engineering

Quartile 4 - 2016/2017

4SC020

Embedded Motion Control

Initial design document

Coordinator:
dr.ir. M.J.G. van de Molengraft
prof.dr. H.P.J. Bruyninckx

Group 4:

M.R.M. Beurskens	0843080
N.H.J. Janssen	0842075
M.C.W. Schouten	0869793
J.A.E. Verest	0844791
M.S. de Wildt	0776941
R.H.M. Winters	0854048

Technische Universiteit Eindhoven University of Technology

Where innovation starts

Eindhoven, May 10, 2017

1 Introduction

In this design document the initial design of a maze solving robot is specified in terms of requirements, specifications, components, functions and interfaces. It is implemented on the PICO hardware platform. A task-skill-motion framework is used in order to describe the robot behaviour. This will also be implemented in C++.

2 Requirements and specifications

PICO is required to autonomously navigate through a maze and find the exit without bumping into any of the walls. A door will be present in the maze, and must be opened by "ringing the bell" in a suitable location whilst standing still. Open spaces might be present, and the walls are approximately axis aligned. The maze might also feature loops. Therefore PICO is required to do the following:

- Recognize walls and corridors in a robust manner. The walls are axis aligned, however not perfectly. PICO should allow for an alignment error. It also has to stay away from the walls with a sufficient margin (e.g. 10 cm between robot and wall). Bumping into a wall will count as a failed attempt.
- Identify possible "door areas" in a robust manner. "The door will be situated at a dead end, a dead end being defined as a piece of wall with minimum length 0.5 meter and maximum length 1.5 meter with side-walls of at least 0.3 meter length [1]". Alignment errors again need to be taken into account.
- Map the maze to determine whether the area has been visited and to build a world model. Odometry and a Laser Range Finder (LRF) are used to build a world map and determine the location in the world. The odometry and LRF data should be synchronized to avoid drift in world location and mapping (SLAM).
- Recognize corners as nodes in order to build an abstract and simplified maze representation consisting of nodes and lines.
- Cary out a maze solving strategy based on its world model and solve and execute corresponding routing. The solving routine and the routing should be able to handle loops and open spaces and are based on the simplified maze representation. This enables more general strategy composition and faster computation.
- Open doors by ringing a bell in the door area. No part of PICO should be located outside the 1.3 meter range of the door and PICO should be standing still whilst ringing the bell. The door opens completely within 5 seconds after PICOs request. Once the door is completely open, PICO needs to pass the open doorway to find the exit of the maze.
- The software should be easy to set up and deploy, i.e. using git pull and with a single executable.
- Clear the maze as soon as possible, with a total available time of seven minutes for two attempts. A maximum speed of $0.5m/s$ translational and $1.2rad/s$ rotational is given.
- Do not stand still for more than thirty seconds when inside the maze.
- Recognize when the maze is solved.

3 Components

Pico features a Laser Range Finder (LRF) to detect surrounding obstacles and wheel encoders to measure odometry data. The wheel base is holonomic, enabling it to turn on the spot and drive in any direction. An emergency button is present on the remote. PICO runs on batteries and should always be charging when not driving around.
The software runs on an Intel i7 processor running Ubuntu 14.04 LTS. The Robotic Operating System (ROS) is used as a framework for PICOs software. However, an intermediate software layer is provided which can be used to access sensor data and actuate the robot. Our own software layer is divided into five main components: Actuation, sensing, world model, task manager and skill.

4 Functions

- **Configuration**: Defines a number of parameters. This is a header file so no inputs or outputs.

Actuation

- **Move**: This function calculates the translational velocity in x and y direction and the rotational velocity corresponding to the direction vector and net velocity outputted by the path planning function. The calculated velocities are sent tot the base actuators. *Input:* Direction vector, total velocity, *Output:* -.
- **Stop**: Stops the robot by sending the base velocities of magnitude zero. *Input:*-, *Output:*-.
- **Rotate**: Rotates the robot 90 or 180 degrees. First checks the robot has enough space to rotate without hitting the wall. If this is not the case, first a call to Move is made to move the robot away from the wall.
- **Open door**: Makes sure the robot stands still, rings the bell to open the door and waits until the door is opened. *Input:* LRF data, *Output:* -.

Sensing

- **Get LRF data**: Gets the data from the laser range finder. *Input*: -, *Output*: LRF data
- **Get odometry data**: Gets position data from omni-wheels. *Input*: -, *Output*: odometry data

World Model

- **SLAM**: Simultaneous Localization And Mapping, this updates the position of PICO in the world model using new sensor data. *Input*: LRF data, Odometry data, *Output*: World model
- **Object recognition**: This function will estimate which objects there are in the World Model using the geometries of the environment. *Input*: World model, *Output*: Objects with their coordinates

Task Manager

- **Task monitor**: Determines which piece of code will be executed next.
  Input: -, *Output*: -

Skill

- **Filter data**: Moving average filter to minimize noise.
  Input: noisy data, *Output*: averaged data.
- **Finish**: Recognizes that the finish line has been crossed and that Pico can stop searching for the exit. The main script can be terminated.
  Input: World model, *Output*: -
- **Strategy**: Gives a general direction based on the maze solving algorithm that is chosen.
  Input: World model data, LRF data. *Output*: Target point.
- **Path planning**: Uses an Artificial Potential Field to output a direction vector considering the target point outputted by the strategy function and the LRF data.
  Input: Target point, LRF data. *Output:* Direction vector

5 Interfaces

The software functions have multiple inputs and outputs and can be grouped by functionality. If data from the sensing component is used by a function from the skill component, the data is transferred between these components, this is referred to as an interface. A rough overview of the interfaces is shown in Figure 5.1. The dashed line represents that the actions of the robot will influence what the sensors measure.

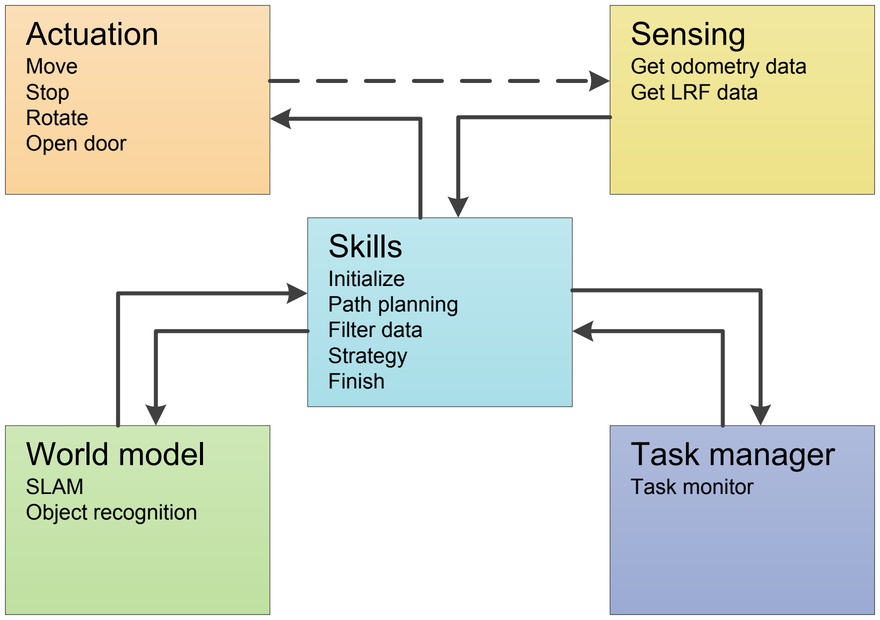

Figure 5.1: Software architecture

References

[1] *Maze Competition*, CST Wiki, http://cstwiki.wtb.tue.nl/index.php?title=Embedded_Motion_Control#Getting_Started , 2017