Department of Mechanical Engineering

4SC020 – Mobile Robot Control (Embedded motion control)

Design Document

Name	Student number
D. van Boven	0780958
M. Katzmann	1396846
R. Konings	1394819
B. Kool	1387391
R.O.B. Stiemsma	0852884
A.S.H. Vinjarapu	1502859

Tutor: Marzieh Dolatabadi Farahani

Eindhoven, May 4, 2020

Introduction

This document describes the design of a PICO application. A PICO is a robot that can monitor the enviroment by sensors. There are two competitions for the robot to complete. The escape room competition, where the robot needs to escape out of a rectangular room, without bumping into a wall. And the robot needs to complete the hospital competition, where the robot needs to maneuver through a hospital, where it has to avoid obstacles.

Requirements

A requirement tree has been set up with the requirements from the stakeholders of the project. The stakeholders are the customer, the hospital employees and the developers of the robot. From here on, the environmental requirements, indicated with the orange boxes, have been acquired. From these environmental requirements, the border requirements, indicated with the purple boxes, have been acquired. Lastly, the system requirements have been acquired, indicated with the green boxes. Some of the green boxes have a set value assigned, indicated with a blue ball. And other system requirements are out of scope for the project team, indicated with a red square. The requirement tree of the hospital room can be seen in figure 1. The requirement tree of the escape room is indicated with the red dotted line.

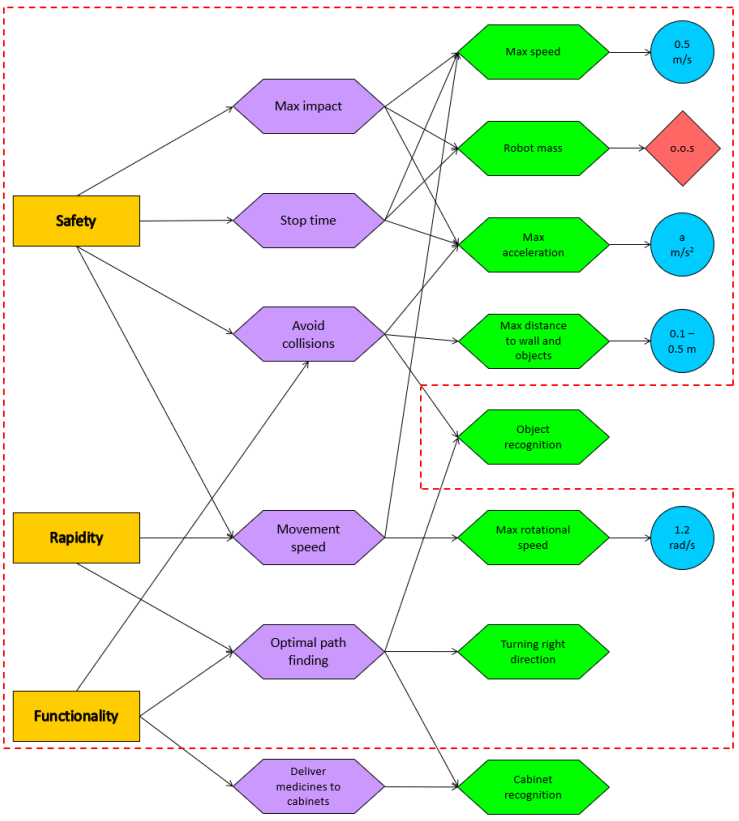

Figure 1: Requirement tree hospital room(dotted area: escape room)

The values assigned to system requirements are still subject to change as the project progresses. In addition, two values have not been chosen yet. Namely the maximum acceleration and the maximum distance to walls and objects. The maximum acceleration will be chosen accordingly once more information of the robot is available. The maximum distance to walls and objects has been chosen between a range from 0.1 and 0.5 meter and will also be chosen accordingly later on when more information is available.

Functions and Interfaces

In this section, the proposed architecture is described by means of its constituent functions and their relationships (interfaces) between one another.

Functions

PICO Interactions

These are all low-level interactions consisting mostly of calls to the IO API (which represents communication with PICO).

- **actuate(v, u, th).** Casts a desired velocity profile $[\dot{x}, \dot{y}, \dot{theta}]$ to the robot.
- **getLRF().** Pulls LRF data from the robot. Returns a `Laserdata` struct.
- **getODO().** Pulls odometry data from the robot. Returns an `OdometryData` struct.

Mapping

This section describes the outlines of the internal map model and its learning process.

- **seeLocal(Laserdata).** Interprets laserdata as a local vision envelope, and stores the outcome locally.
- **loc2glob().** Transforms local sensor data into a global perspective using calibration elements (e.g. intersections of wall lines).
- **tf2map().** Transforms sensor data (in a global coordinate system) to `Mapdata`, which involves rasterizing the received data.
- **integrate().** Compares and contrasts new data with old data, and modifies (and/or expands) the global map model accordingly.

Navigation

This section covers the system functionalities responsible for (intelligent) navigation. All functions here have read access to the world model's internal map and LRF envelope.

- **localize().** Deducts the PICO's current location in the map model based on its most recent LRF data.
- **pathfind(x, y).** Employs a pathfinding algorithm to find a combination of rotations and straight motions (longitudinal *or* sideways) that will bring the PICO from its current location to the global coordinate (x, y). Returns a `PathData` struct which describes the intended locations, and necessary rotations and local (x, y) motions to reach them.
- **nextTrajectory().** Takes the next first data from `PathData`[1], and calculates the necessary velocity profiles and timings to achieve the desired motion smoothly. Returns this as a `Trajectory` struct.
- **compensate().** Checks the PICO's current location compared to its intended location within the planned path, and determines whether compensation is needed in the sense of adjusting the path, the PICO's position, or recomputing the path.

Supervisor

This section covers the governing control logic of the proposed system. The idea is that the supervisor makes choices, tunes parameters, and controls flow-of-command within remaining code depending on its mode, and changes modes dynamically. Modes are described here as functions, but may end up in a different form during implementation.

[1] This pops the corresponding entry out of `PathData`

- **init().** Initializes the robot, checks (as far as possible) whether safe operation is possible. Sets the mode to *scout*.
- **scout().** PICO is uncertain about its surroundings, and hence scans its surroundings, moving as little as possible in doing so. In this mode, safety parameters are conservative. Once a useful feature (such as a door) is found, or map confidence improves enough for PICO to feel safe, control is yielded to the *move* mode. If all exploration options are exhausted and PICO is still uncertain, control is instead yielded to *wait*.
- **move().** PICO knows enough, and knows where it wants to go. This mode sets safety margins much more aggressively than *scout* mode, and looks to move long, efficient motions in pursuit of PICO's intended destination. During motion, PICO keeps updating its map, and reverts to *scout* mode if certainty dips to unacceptable levels.
- **wait().** PICO decides its goal is currently impossible. Waits, and occasionally returns to *scout* mode to see if the situation has changed[2].
- **act().** An empty token mode to represent PICO arriving at a destination and doing something there.

Interfaces

Keeping the aforementioned functionalities in mind, and given that the modes of the *Supervisor* module are represented here as separate locations for clarity's sake, a rough depiction of the proposed system structure is shown in Figure 2 below.

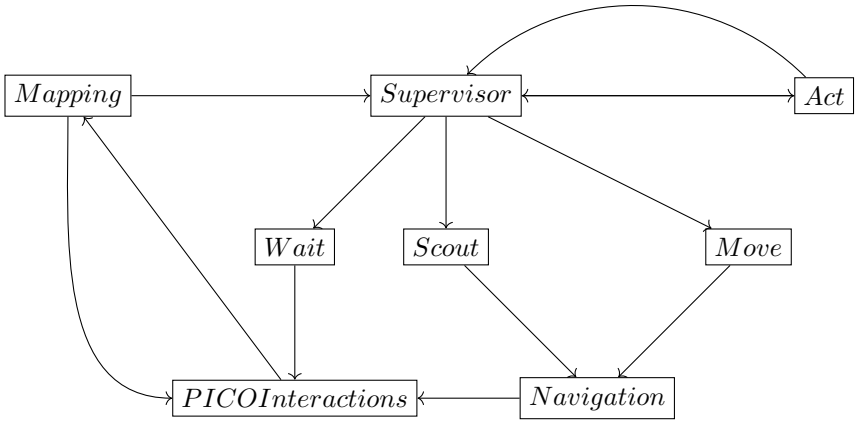

Figure 2: State Machine Representation

[2]This covers the test case where a target location is barricaded.

Components

The hardware of the PICO robot used to smoothly execute the functions mentioned above, consists of the following components:

- Sensors
  - Laser Range Finder
  - Wheel Encoders
- Actuators
  - Holonomic wheels
- Computer
  - Intel Core i7 Processor and 8+ GB of DDR3 RAM

Specifications

System: PICO

1. Maximum translational speed: 0.5 m/s
2. Maximum rotational speed: 1.2 rad/s
3. Wheel encoders used for odometry
4. Laser Range Finder(LRF) used for distance measurement
   - Measurable angle: 4 radians.
   - Resolution: 0.004 radians.

Environment:

1. Escape Room Challenge:
   - Shape of the room is rectangular.
   - PICO starts at a random position.
   - Width of the corridor is between $0.5m$ to $1.5m$.
   - Finish line is more than $3m$ into the corridor whose orientation is perpendicular to the wall.
   - Walls are not necessarily perfectly straight.
   - Corners are not necessarily perfectly perpendicular.
   - Challenge is to be finished in 5 mins.
2. Hospital Challenge:
   - Hallway is approximately $1.5m$ wide and is not necessarily straight.
   - Doors in the hospital are 0.5m to 1m wide.(closed or open).
   - A random number of static and dynamic obstacles will be present throughout the hospital.
   - Challenge is to be finished in 5 mins.