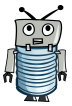
Marlene Böhmer, s9meboeh@stud.uni-saarland.de, 2547718
Maximilian Köhl, s8makoeh@stud.uni-saarland.de, 2553525
Maximilian Schwenger, schwenger@stud.uni-saarland.de, 2542438
Ben Wiederhake, s9bewied@stud.uni-saarland.de, 2541266



1 E-Puck not as specified

We are short on time as we had to deal with a few problems with the E-Puck library and E-Puck specification:

- Proximity sensors do not yield values as specified. We had to measure the raw values, come up with a reasonable approximating formula.
- The Bluetooth handler of the E-Puck firmware does not have a nice way to prevent packet loss, and does not support packages over 64 bytes (which is far too low for our use case).
- The E-Puck firmware uses several different interrupts, and it was highly unclear which interrupt timers were used by the firmware or free for usage by us.

Our approach was to create our own firmware, which proves to be more usable and less error-prone than the pre-existing one.

2 T2T communication

Technically, Tin-Bot-to-Tin-Bot-communication only improves the global completeness of internal maps, despite being a major technical effort. Therefore, this feature is *not* present in our prototype. Thus, we will present a *single*[1] working Tin Bot.
Although T2T is already planned for and in some parts already specified, our approach is to implement T2T last. However, of course we already use Bluetooth to exchange debugging data and the internal map, so we effectively *are* implementing T2T as we go anyway.

3 Rescue mode

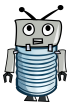
We only have two options for presentation: either we demo the Right-Hand-Follower (which definitely works), or we demo the overall system, which would require *all* the components to be perfectly working. At least currently, the next stage does not work yet (the `victim_direction` module needs working IR sensors; see below). We cannot guarantee the latter for the prototype yet, so we only demo the rescue mode RHR.
For the final product, our approach is to get all components working.

4 Moving the victim

Although our current prototype of the victim is very light and offers very little friction against the ground, the Tin Bot cannot turn while the victim is attached. This is bad, as our atomic path-following block first turns and then moves.
For the final product, our approach is to solve this by any of the following:

- making the victim even more light-weight, maybe use wheels

[1] thus we also do not need any synchronization that makes sure that precisely one Tin Bot rescues the victim



- changing the turning building-block to move forward-backward in order to turn; not unlike a car
- using another E-Puck for the victim, which then actively follows the “rescuer”

5 IR sensors (victim detection)

Our IR sensors do not work as well as expected. They are highly sensitive to noise and reflections, contrary to our earlier experiments with own hardware.
This seems to be also due to the IR emitters being highly direction dependent instead of “diffuse” in all directions.
Our approach is to sacrifice angular and temporal resolution in return for stability of the received signal.

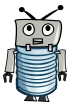
6 What *does* work

Right-Hand-Following definitely works, so this is part of the prototype. The LPS also works, including image recognition, conversion to the required units and sending the data via Bluetooth. As Right-Hand-Following does not make use of this information, the LPS is demonstrated through a web interface instead. Finally, the hardware to determine whether we are currently connected to the victim works reliably, and the end-to-end I2C communication works: the highly noisy IR data is available to the Tin Bot software.

7 What is implemented but not presented

Most internal logic and path finding is already implemented and unit-tested. However, there is no really visible intermediate step between Right-Hand-Following and truly solving the maze, so we cannot showcase the internal logic, i.e., all the state machines in the "Controller", as per virtual Prototype.
Our approach for the Prototype Session will be to try and get certain parts working in stand-alone mode anyway.

Marlene Böhmer, s9meboeh@stud.uni-saarland.de, 2547718
Maximilian Köhl, s8makoeh@stud.uni-saarland.de, 2553525
Maximilian Schwenger, schwenger@stud.uni-saarland.de, 2542438
Ben Wiederhake, s9bewied@stud.uni-saarland.de, 2541266



1. Overview

After a major catastrophe a brave team of robots is sent out to save a damsel in distress. The collapsed building is simulated by a maze in which the victim, represented by a teddy bear, waits for it's rescue. The maze is laid out on a table, the walls are built out of paper. Some of the walls contain holes as windows or broken down beams of wood or steel. Will they save the day?

2. Functional Requirements

A. Must-Have

MR1 while the Tin Bot is operational, a green LED shall be on

MR2 the "LPS" (local positioning system) needs to supply the Tin Bots with location and orientation data via Bluetooth

MR3 the Tin Bots must share information about the victim

MR4 chart the inspected environment using the given LPS signal

MR5 find the victim by...

a) ... exhaustive search, if nothing about the victim's location is known

b) ... attempted shortest path, if information about the victim's location is available

MR6 grab the victim using magnets after finding them

MR7 use proximity sensors / IR receivers to detect whether victim is grabbed correctly

MR8 move the victim out

MR9 while the Tin Bot is escorting the victim, the front red LED shall be flashing

MR10 use the IR sensors to pick up signals from the victim

MR11 ~~detect and stay within the borders of the table~~ (not possible because we need to connect our own extension board to the connector of the ground sensor board)

MR12 the surrounding 8 red LEDs shall represent which/whether the IR sensors receive some signal

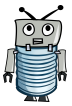
B. Nice-to-Have

NR1 be able to behave reasonably in terms of continuing the search if the LPS stops working as intended

NR2 export the map

NR3 be able to behave reasonably if some of the Tin Bots go defunct

NR4 be able to behave reasonably if the map changes



NR5 detect if there is no victim

C. Must-not-Have

N1 use actual GPS

N2 use actual gyroscope

N3 use camera to identify the victim

N4 deal with unsolvable mazes

N5 deal with more than one victim

N6 try to remove obstacles with Tin Bots

N7 try to detect / react to a malfunctioning / byzantine LPS (although a certain tolerance must be respected)

N8 secure against any kind of attack (physical or non physical)

3. Non-Functional Requirements

A. Must-Have

MR13 must not be slower than worst-case brute force

MR14 if there is information do not just run to the position where the information was gathered but instead try to actually localize and find the victim efficiently

MR15 avoid any collisions except with the victim

MR16 do not make any assumptions about the maze except being solvable

MR17 the exits of the building are the set of the Tin Bots' starting positions

MR18 Tin Bots are equipped with a visual marker on top

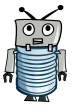
MR19 the custom extension boards (with IR sensors and ATmega328p) are easily deployable / removable

MR20 sampling sensor information from IR-Sensors in a real time manner using the ATmega328p

MR21 pick up received signals from ATmega328p in a real time manner such that no information is lost by being overwritten

MR22 transmitting pre-specified time-critical signal sequence using the IR-emitters using the ATtiny2313

MR23 each ATmega328p must be aware of the color it is "wearing" (for self-identification via the LPS)



B. Nice-to-Have

NR6 be gentle to the victim, i.e. do not knock them over or squeeze against a wall

NR7 use the shortest known path out of the building (see starting positions)

C. Must-not-Have

N9 deal with uneven floors or multiple floor buildings

N10 use actual collapsed buildings or victims

N11 obstacles that are not detected by the proximity sensors (e.g., walls that are too small)

N12 deal with more than 8 Tin Bots

N13 allow Tin Bots to *come online* while the overall system is already active

4. Use-Cases

A. System Startup

primary actor: user

goal in context: activate the Tin Bot(s)

precondition: LPS is up and running; E-Puck-batteries are connected and charged up; Tin Bot power switches are in the off position; our custom extension boards on the E-Pucks are equipped and clearly visible from the LPS; the custom extension boards have pairwise distinct colors (for identification by the LPS)

trigger: power switch of one or more Tin Bots is switched on

scenario:

1. user places Tin Bots on surface
2. user switches on the Tin Bots

exceptions: Tin Bots are too far away; LPS is too far away; any Tin Bot cannot properly move on its own (e.g. placed upside-down)

B. System Shutdown

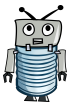
primary actor: user

goal in context: deactivate the simulation / system

precondition: at least one Tin Bot's power switch is in the on position

trigger: power switches of the currently-switched-on Tin Bots are switched off

scenario:



1. user switches off all of the running Tin Bots
2. user switches off the LPS

C. Removing A Tin Bot

primary actor: user

goal in context: simulate hardware failure in one of the Tin Bots

precondition: at least two Tin Bots one the table are switched on and running; requirement **NR3** is implemented

trigger: power switch of one of the switched-on Tin Bots is switched off

scenario:

1. user selects some Tin Bots to be deactivated (at most all except one)
2. user switches off these Tin Bots
3. user removes none, some, or all of these Tin Bots (the other switched-off Tin Bots shall be considered "obstacles")

exceptions: only one switched-on Tin Bot remains; no Tin Bots are running; switching off and not removing any of the selected Tin Bots would render the maze unsolvable

D. Adding/Removing Walls

primary actor: user

goal in context: simulate walls segment that collapse or break away

precondition: requirement **NR4** is implemented; E-Pucks are running; if removing a wall segment: at least one "wall"-segment on the table acting as a wall segment

trigger: user manually removes or adds one or more of the wall segment

scenario:

1. user removes some wall segment
2. user places the removed wall segment somewhere else

exceptions: adding a wall must not render the maze unsolvable

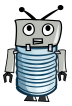
E. Disabling LPS

primary actor: user

goal in context: simulate connectivity problems between LPS and Tin Bots

precondition: LPS is up and running; requirement **NR1** is implemented

trigger: user switches off the LPS (either by software, or by switching off the Raspberry Pi, or by blocking the camera's view)



scenario:

1. user literally switches off the Raspberry Pi

F. Re-enabling LPS

primary actor: user

goal in context: simulate LPS coming back up again (see previous use case)

precondition: LPS is not running

trigger: user undoes the action from the disabling step

scenario:

1. user switches the Raspberry Pi back on

5. Environment properties

A. Plant

E1 victim placed in maze, at least 1cm from the wall (so magnets don't accidentally trigger through the wall)

E2 victim sending signals (using IR emitters)

E3 victim wears magnetic belt

E4 solvable maze (including table and walls; walls need to be of sufficient height to prevent signals radiating over the walls)

E5 flat surface

E6 unblocked view from the camera to the maze

E7 no items in the maze that have a bright blue or bright red color

E8 the LPS knows in advance how many Tin Bots will initially be available (see N13)

B. Sensors

S1 power switches

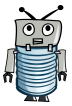
S2 IR receivers

S3 proximity sensors

S4 Bluetooth receiver

S5 camera (LPS)s

S6 ground sensors



C. Actuators

A1 Bluetooth emitters

A2 motors (two per E-Puck)

A3 IR emitters (victim)

A4 LEDs (E-Pucks)

6. Platform

A. Hardware

H1 at least two E-Pucks

H2 ATmega328p (to connect IR sensor with E-Pucks)

H3 IR sensors (for E-Pucks)

H4 magnets (for E-Pucks)

H5 Raspberry Pi 3 (for LPS)

H6 camera (including stand) (for LPS)

H7 teddy (a.k.a. victim)

H8 magnet belt (for teddy)

H9 IR emitter (for teddy)

H10 ATtiny2313 (to put unique signal on IR emitter)

B. Libraries

L1 E-Puck base libraries

L2 Raspberry Pi base system (Linux, . . .)

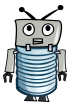
L3 Raspberry Pi Camera module

L4 OpenCV or similar (for image analysis)

L5 base libraries for ATmega328p and ATtiny2313

7. Amendments

- Remove MR11 (don't fall off the table)
- Include distance in E1 (magnets vs. walls)
- Modify use-case E to include "camera blocked" as a trigger



- Add requirement E7 that there are no similar colored things that confuse out image recognition
- Add requirement E8 that Tin Bots never come online while the system is already active (e.g., there already is map data that would need distribution)