Real Time Operating Systems

Final Report

Wojciech Juszczak
Jarosław Szumega
Adam Superczyński

28.05.2018

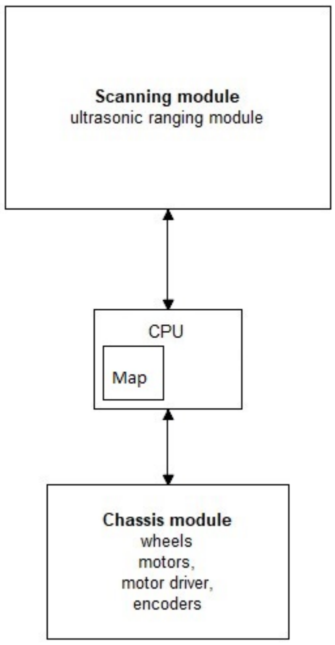
Contents

Chapter 1

Introduction

The aim of the project was building automatic, self-driving robot with ability to detect and avoid obstacles. System was implemented on STM32 with the use of FreeRTOS.
General architecture diagram of project is presented below.

Figure 1.1: Architecture diagram



The whole architecture can be divided into three elements:

- Scanning module - used for surrounding scanning and informing about distance to obstacle,
- Map - information about surrounding stored inside CPU memory. Based on it robot makes decision about next movement,
- Chasis module - deciding about movement, controlling motors ...

Chapter 2

Detailed description of project

2.1 Scanning module

For this purpose proximity sensor HCSR-04 is used. To start measuring it needs 10us pulse on pin TRIG. After that it will response with pulse on ECHO pin which has width proportional to the distance. High priotity task activates measurement by clearing the timer and setting high signal on TRIG pin. Task becomes blocked and waits for interrupt from ISR. After 10us there is timer interrupt in which signal on TRIG pin is set to low. Next, values of timer in rising and falling edges are saved. On falling edge notification is sent to task. After receiving notification distance is calculated and saved to mutex - locked global variable. Task is blocked for 100ms before next measurement.

2.2 Map for constrained memory limit

2.2.1 Problem statement

There is a necessity of having map stored in the robot memory. There is a few reasons of such an approach:

- saving the scanning data aquired from sensors,
- having a structure that is easy to send and interpret on another device (e.g. for rendering 2D map on PC),
- introducing safety mechanisms (robot aware of obstacle ahead of it).

The assumption was to have a map constructed of tiles (where one tile is more or less the surface covered by the robot – 30cm x 30cm).

The problem was the storage – if we need to cover room 10m x 10m, there is a need of having around 1200 cells representing each "tile", while the status of tile can one of four defined:

- unknown,
- scanned and marked as obstacle,
- scanned and marked as free,
- marked as robot position.

Using the array of integers is a waste of memory (which is also very limited in the arm processor).

2.2.2 Solution

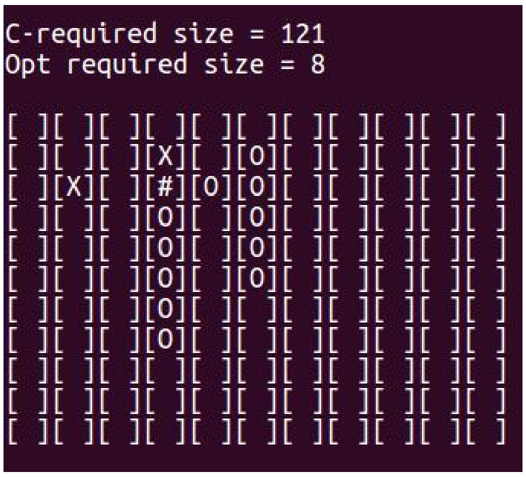
To optimize the map, there was developed optimized C–code for memory access and manipulation.
It takes advantage of bitwise operations, what drastically saves space (up to 16 times less memory is occupied by the map). There are four states, what means that each can be represented using only 2 bits in entire integer, therefore in 32–bit representation we get 16 states saved in one bit, in 16–bit, there can be stored 8 states, etc.

In the files **robot_map.h**, **robot_map.c** there are all necessary functions that were designed to deal with this optimized map storage (interpreted one–dimensional array as two–dimensional map where each integer is seen as **sizeof(int)*8/2** cells for storing states).

Currently there is support for:

- allocating map and basic operations on it,
- get/set the individual cells,
- setting n–cells ahead free, setting next cell free, setting next cell blocked, setting cell blocked at distance X,
- answering if the next cell is free (for robot safe movement),
- marking robot movements such as going forward, turning left, right and backwards.

There also a useful print function prepared for visual interpretation of optimized map – it was used mainly for testing, but can be also an interface on PC computer to show the map.
It is simple and requires only knowledge of rows, columns and array – therefore the data transfer is also very simple.



```
C-required size = 121
Opt required size = 8

[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][X][ ][O][ ][ ][ ][ ][ ]
[ ][X][ ][#][O][O][ ][ ][ ][ ][ ]
[ ][ ][ ][O][ ][O][ ][ ][ ][ ][ ]
[ ][ ][ ][O][ ][O][ ][ ][ ][ ][ ]
[ ][ ][ ][O][ ][O][ ][ ][ ][ ][ ]
[ ][ ][ ][O][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][O][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
```

Figure 2.1: Example of map 11x11 cells
(empty – unknown, X – obstacle, O – free space, – rover position)