Robotics project

Lukas Rustler, Martin Sramek, Cemre Eren

January 2020

1 Description

This is project made as final project for ISR lab. The task of the robot is to navigate through field, detect obstacle, map the field and find a red spot somewhere in the field. Example of the field can be seen in Figure 1 below and the whole assignment is described in assignment pdf.

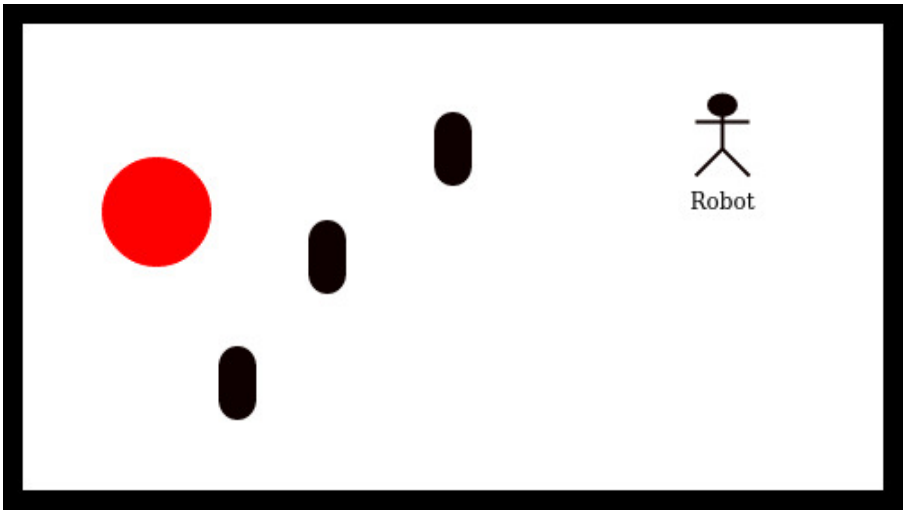

Figure 1: Example of world

As robot was used Lego EV3 robot with Debian linux. And for simulator we have chosen the VREP simulator.

2 Artificial intelligence

2.1 Map

The whole map is divided into small squares, with given resolution. In our case we divided map in simulator, which is 1x1 meter, to 10x10 field. Which means, that one "square" is 10cm. This can be easily changed, as we have added special variable for this.

2.2 Robot behaviour

The first thing the robot does is to follow line all around the playing field, which is then used as boundary for it. After it reaches the place where it started to follow the line it corrects its position to look "north" (this is done by moving until it reaches black line on the right side and then move a little bit left). Now it moves forward and every time it bumps into black line it turns to one column left.

As robot always going north or south, it can check if it is in right column. This is used to correct its straight movement. Every time it moves outside current column, it starts to correct its position by moving left/right. This keeps it straight even if its 90°rotation are not precise.

If it bumps into an obstacle, it moves a little bit backward and then makes rotation to the safe position. Safe position is usually on the right side, as it has been already mapped by the robot. After getting around the obstacle, it moves back to previous column and continues wandering.

If the robot founds a red spot, it stops its movement and program is waiting for user input to end.

2.3 Virtual and real robot

Both of the robots have its own class with defined functions. We defined the functions the same for both robots, in sake of easy turning from simulated to real robot. The functions are:

- move_forward(speed, time) - moves robot forward.
- move_backward(speed, time) - moves robot backward.
- rotate_right(speed, time) - rotates robot right, on place.
- rotate_left(speed, time) - rotates robot left, on place.
- stop() - stops the robot.
- color() - returns color which is being seen by the color sensor. In intensity from 0 to 100.
- touch_right(), touch_left() - returns state of the right/left touch sensor. As 1 (pressed), 0(non-pressed)
- position() - returns the position of the robot. Array $[x, y]$, in metres.

3 Prolog

Prolog is used for the main state machine during robots wandering through the map. The behaviour is described below.

```
upper_corner(0).
is_up :- upper_corner(C), x_pos(X), C=:=X.
is_down :- bottom_corner(C), x_pos(X), C=:=X.
is_left :- column(C), y_pos(Y), Y<C.
is_right :- column(C), y_pos(Y), Y>C.
x_pos(X) :- position(A), [X|_]=A.
y_pos(Y) :- position(A), [_|Z]=A, [Y|_]=Z.

stateMachine(state1) :- obs(X), X==1, !.
stateMachine(state2) :- is_up, !.
stateMachine(state3) :- is_down, !.
stateMachine(state4) :- is_right, !.
stateMachine(state5) :- is_left, !.
stateMachine(state6) :- !.
```

The main Python program is communicating with Prolog interpreter with the use of PySwip bridge. The position of the robot is dynamically asserted into Prolog knowledge base and Prolog is returning the proper state for state machine.

4 Results

4.1 Simulated robot

The simulation works well and met conditions of the assignment. The only problem is that the simulation is very depending on the CPU processing power. It acts differently on different computer and also in acts differently on the same computer based on running processes etc. To solve this problem we added an CPU constant, which can be easily edited and it change speed of all movements, so it takes just few tries to get simulation working. But most probably the simulation will not run perfectly on the first try on another computer. Otherwise the simulation is good and resulted video can be seen on this video.

4.2 Real robot

With real robot more problems occurred:

- We were not able to install Prolog interpreter
  - There is no install candidate on PPA for Debian linux on EV3 robot
  - Manual building also failed, because the robot is very slow and the building took forever and never ended
  - We rewrote the Prolog into Python for real robot
- The odometry on robot is really inaccurate
  - The odometry is working on short, straight distance
  - Motors are really inaccurate, they return different driven angles than what are really driven
  - For example, motors returned 150°but in fact they drove 140°. We have wheels with diameter of 5.5cm, so the error for this is:

$$circumference = 5.5 \cdot \pi = 17.27cm \tag{1}$$

$$distance\_sensor = circumference \cdot rotations = 17.27 \cdot (\frac{150}{360}) = 7.195 \tag{2}$$

$$distance\_real = 17.27 \cdot (\frac{140}{360}) = 6.716 \tag{3}$$

$$difference = 7.195 - 6.716 = 0.48 \tag{4}$$

    We can see that the difference is almost half centimeter, which adds up to big difference in the end
  - So the real robot is not as responsive as the simulated one and sometimes makes mistakes
- Color sensor returned strange values sometimes
  - We use mode when color is returned as number from 0 to 100 as intensity of reflected light (black reflects less light than white)
  - sometime the robot can see a black on white and turns illogically
  - we put the color sensor closer to the ground, which helped to shield the shade of the robot, but it still recognize a bad color sometimes
- Our created field was not really created perfectly
  - Out playing field is just really basic
  - we did not have a black tape, so the borders are green
  - also we did not have another color, so the "red" point is just more green on itself, which creates a darker (almost black) color
  - also as obstacle a water bottle is used, but it needs to be held when robot bumps into it or it moves

With respect to problems mentioned above can make mistakes sometimes. But he is able to make the course, as can be seen in this video.

4.3 GUI

We have implemented also a GUI which shows the current map made by the robot. It works quite well in simulation, only sometimes robot does not map some position. But it could be fixed with higher resolution.

Problems occurred with the real robot. The CPU of the robot is very slow, so it haven't been able to send data to computer without loss of speed. Also Bluetooth on the robot is very unstable and the connection got dropped very often. From this reason we decided not to implement this for the real robot.

5 Summary and future work

We have successfully created and artificial intelligence for robotic system, which can make a map and avoid obstacles. We verified it in simulation and also on the real robot. In the future we could focus on these problems:

- Install Prolog on the EV3 robot
  - probably build it from sources
- Test the motors and find a way to avoid odometry problems
  - We need a lot more time to test motors in different scenarios to compute errors
  - if we have more data we can invent some way to overcome inaccuracies
    - with some correcting constants or maybe neural network trained for this problem
- Create a better field
  - It would surely help if we have better map
  - probably printed on big paper with proper colors etc.
- Find another way how to get data from robot to computer to display the GUI
  - Another technology than Bluetooth or find a way how to speed up the