Following a line using the camera of the e-puck robot
Course project

Alexis Baron, Cyril Wendl, Caroline Menard

30 mai 2016
Table of contents

1. Introduction

2. Methodology

3. Implementation
   3.1 Line following (ST_FOLLOWLINE)
   3.2 Obstacle avoidance (ST_LOVER)
   3.3 Line finding (ST_FIND)
   3.4 Interaction between the states

4. Robustness and portability

5. Conclusion
Introduction: Goals

- Follow line
- Avoid obstacles
- Find line

- ...For different types of lines
- In real and simulated environment
- With different robots
Introduction: Sensors

- Camera
  - Simulation: 52 × 39 RGB
  - Real: 640 × 480 panchromatic, downsampled to 40 × 40
- 8 IR sensors, used for proximity measurements
"A (finite) state machine is a model of system behaviour which can consist of states, state transitions and actions." [1]

1. Reading sensor values (proximity and camera), being updated continuously in an infinite while-loop;

2. Determination of the e-puck’s state depending on the sensor values;

3. Execution of the part of code belonging to each state.
Implementation strategy: state machine

Figure 1: State machine
Line following (ST_FOLLOWLINE)
In the simulation: simply R=G=B=0

In reality: fluctuation values, coded on 8 bits \(2^8 = 256\) possible values, ranging from 0 to 255), perfect white or black only in case of dysfunctional pixel

→ Threshold value for black / white pixels
Line following: Camera values

Figure 2: E-puck camera output
Line following

- Bottom 5 lines used (reduce fluctuations / noise)
- Differential wheel speeds depending on the number of black pixels on either side

**Listing 1: Line following**

```c
    case ST_FOLLOWLINE:
        dp=((double)c_l_5)/((double)c_r_5+(double)c_l_5+1.0); // check if more black pixels left or right
        printf("DP: %f\n",dp);
        if(dp>0.5){// more left pixels, turn left
            left_speed = NORM_SPEED+NORM_SPEED*(1-dp);
            right_speed = 2*NORM_SPEED-NORM_SPEED*(1-dp);
        }
        else{ // more right pixels, turn right
            left_speed = 2*NORM_SPEED-NORM_SPEED*dp;
            right_speed = NORM_SPEED+NORM_SPEED*dp;
        }
        follower++;
        break;
```
Obstacle avoidance (ST_LOVER)
Proximity sensors

![E-puck IR sensors](moodle.epfl.ch)

**Figure 3:** E-puck IR sensors (moodle.epfl.ch)

**Listing 2:** Proximity sensor weights

```plaintext
2 prox_right = (3*prox[0] + 5*prox[1] + prox[2])/9;
```
Obstacle avoidance: implementation

Listing 3: Obstacle avoidance

```c
if(prox_right>prox_left){//obstacle on the right side
    ds = NORM_SPEED * (prox_right / THRESH_PROX);
    left_speed = 2*NORM_SPEED - ds;
    right_speed = ds;
}
```

Introduction Methodology Implementation Robustness and portability Conclusion
Line finding (ST_FIND)
Line finding: implementation

Listing 4: Line finding

```c
    case ST_FIND:
        ...
        if(prox_right>prox_left)//turn left
            ds = NORM_SPEED * (prox_right / THRESH_PROX);
            left_speed = NORM_SPEED - ds;
            right_speed = NORM_SPEED + ds;
        }
        ...
    }
```

Figure 4: Find behaviour [2]
Interaction between the states
Listing 5: Interaction between the states

```c
// Determine states
// if line is seen (5 last pixel lines) and no obstacle, follow line
if((c_l_5>LINEPIX_MIN || c_r_5>LINEPIX_MIN) && (c_l_5<LINEPIX_MAX && c_r_5<LINEPIX_MAX)){// line detected
    state=ST_FOLLOWLINE;
    count_lover = 0;
} // if obstacle detected, go around it for at least 70 iterations
else if((prox_right>TRESH_ST_LOVER || prox_left>TRESH_ST_LOVER) && count_lover<70){// obstacle detected
    state=ST_LOVER;
}
else if (count_lover>70){
    state=ST_FIND;
}
```

Introduction Methodology Implementation Robustness and portability Conclusion
Robustness and portability
Robustness and portability: camera

▶ Camera
  ▶ Varying values, according to luminance of the surroundings and e-puck used → Robust threshold value for black / white pixels
  ▶ Actual physical position of the camera in the e-puck
    ▶ Too high / straight: FOV too far away
    ▶ Too tilted: Looks inside its own casing
    ▶ Looking to the left / right changes FOV accordingly, can make it difficult to follow sharp curves in one direction
  ▶ Reading black pixels: robot also detects black pixels if in front of shady object → LINEPIX_MAX

→ Mainly related to the real e-puck
Robustness and portability: camera

(a) Implemented shapes  (b) Shape to improve

Figure 5: Tested line shapes
Robustness and portability: proximity sensors

- Sometimes lag: due to low weight of front sensors (0 and 7)
- Slight differences between both sides left / right despite equal weighting → correction factor
- Possible interference with other emitters of IR, such as incandescent lamp, heat
Further developing

- More rigorous / detailed reporting of success rates
- Test more shapes and more e-pucks
- Calibration of e-pucks via separate script
- Compare used method to Fourier processing of the image
- Use only two front proximity sensors for faster state change
- Initialize state (outside of while loop) to ST_FOLLOWLINE or ST_FIND depending on line recognition
Conclusion

- Good group work
- Important lessons about embedded systems, memory constraints and real-time computing learnt
- Debugging and code isolation as a good strategy for an efficient implementation of the code (→ state machine)
References


Thank you for your attention!