LINE FOLLOWING USING THE EPUCK CAMERA

FINAL PRESENTATION
2ND JUNE 2015
PLAN

1. Introduction
2. Experiments
3. Results
4. Conclusion
INTRODUCTION

- **Purpose of the project:**
  Design a controller (C language), which enables an Epuck to follow a line, thanks to its camera.

- **Defined objectives:**
  1. Follow the line
  2. Search the line
  3. Avoid obstacles
  4. Go around obstacles

- **Two main steps:**
  1. Design the controller on Webots, a high fidelity simulator
  2. Adapt it to work on a real Epuck
Æcole Polytechnique Fédérale de Lausanne

SIS Project, SIE BA6
Sélim Amrari & Baptiste Gex

EXPERIMENTS

- Method based directly on the information given by the camera, i.e. the pixel value.
- This value is between 0 and 255 (grey scale).

Fig1: Initial picture for analysis

Fig2: Grey scale picture
EXPERIMENTS

- With these information, how can I find where to go

```
find_direction()
```

- Count the black pixels on the last line
- If `count > 2`
  - Last line analysis
- If `count < 2`
  - Global bottom analysis
- Calculate `c`, the correction factor
- Set the new left and right speeds

```
wheels_speed()
```
EXPERIMENTS

- If the number of black pixels on the last line of the image is greater than a certain sill, the Last line analysis is executed by the `find_direction()` function.

```c
for (x = 0; x < image_width-1; x++) {
    if (data[image_height-1][x]-data[image_height-1][x+1] > 30) {
        b1 = x;
    }
    if (data[image_height-1][x+1]-data[image_height-1][x] > 30) {
        b2 = x;
    }
}
*corr = ((b1+b2)/2 / (double)image_width);
```

*Correction factor, \( \epsilon \in [0,1] \)*)

*Fig3: Find the border of the line*
EXPERIMENTS

- If the number of black pixels on the last line of the image is under a certain sill, the Global bottom analysis is executed by the `find_direction()` function.

```c
for (x = 0 ; x < image_width ; x++) {
    for (y = 3*image_height/4 ; y < image_height ; y++) {
        if ((data[y][x] < 5) && (x < image_width/2)) {
            count1++;
        } else if ((data[y][x] < 5) && (x > (image_width/2)-1)) {
            count2++;
        }
    }
}
*corr = (double)count2 / ((double)count1 + (double)count2);
```

 Correction factor, $\epsilon \in [0,1]$
EXPERIMENTS

- The correction factor is then sent to the `wheels_speed()` function, which compares this value to the middle of the image, i.e. 0.5. Therefore, we must distinguish 3 cases.

```c
void wheels_speed(double c)
{
    if (c > 0.5) {
        left_speed = MAX_SPEED;
        right_speed = (1.4-c) * MAX_SPEED;
    } else if (c < 0.5) {
        left_speed = (0.4+c) * MAX_SPEED;
        right_speed = MAX_SPEED;
    } else if (c == 0.5) {
        left_speed = MAX_SPEED;
        right_speed = MAX_SPEED;
    }
    wb_differential_wheels_set_speed(left_speed, right_speed);
}
```
# RESULTS

- Code transfer from Webots, to E-puck:

<table>
<thead>
<tr>
<th></th>
<th>Webots</th>
<th>E-puck</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Following line:</strong></td>
<td>Works well</td>
<td>Has random behavior</td>
</tr>
<tr>
<td><strong>Last line analysis</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Searching line:</strong></td>
<td>Works well</td>
<td>Works depending on the light and the distance</td>
</tr>
<tr>
<td><strong>Global bottom Analysis</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Following line:</strong></td>
<td>Not used</td>
<td>Provided a better behavior</td>
</tr>
<tr>
<td><strong>Global bottom Analysis</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Avoiding obstacle</strong></td>
<td>blocks</td>
<td>Works well</td>
</tr>
<tr>
<td><strong>Braitenberg Loop</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RESULTS

- Problems with the code transfer from Webots, to E-puck:
  - Camera setting: different indexation
  - Memory: limited
    - Only one pic 40x40
    - Camera setting
  - Computation system: slower
    - Lost of stability: reset the system
    - Time management wait()
  - Light conditions: much more cases to deal with
    - Random behavior
RESULTS

- Camera settings optimized to:
  - Take only the useful part
  - Use a minimum of memory

- Indexation conversion:
  \[ I = \text{height} - (x+1) \times \text{width} + y \]

---

Fig 4: Useful part of analysis
RESULTS

Control of the process:
RESULTS

- **Last line Analysis:**
  - **Behaviour:** Random
  - **Testing process:** Show many random error and absurd values
  - **Tested alternative:** average analysis on 3 spaced pixel lines for more robustness
  
  Result: not convincing

*Process not strong enough!*
RESULTS

- **Global bottom Analysis:**
  - Behaviour during line searching:
    - Depending on the distance → due to narrow range of view
  - Behaviour during line following:
    - More suitable, more promising
- **Testing process:**
  - Show less random error and absurd value
- **Tested alternative:**
  - Analysis on a lower part of the image for *following line*

*Process much safer for both goal!*
RESULTS

- Which condition determine the mode of analysis?

  → *It depends also on the light conditions:*

  Previous condition:
  Are there blacks pixels on the lower lines?
  -> *not effective*

  New condition:
  Are there enough black pixels on all the image?

*Fig4: Critical situation*
RESULTS

Final solution:

How many black pixels in the picture (representing ¼ of initial range of view) ?

- Searching line:
  Global Bottom analysis on all the image (representing ¼ of initial range of view)

- Following Line:
  Global Bottom analysis localised on the lower half part of the image (representing 1/8 of initial range of view)
CONCLUSION

Following line:
CONCLUSION

Searching the line:
CONCLUSION

- Avoiding obstacles:
THANK YOU FOR YOUR ATTENTION

?? QUESTIONS ??