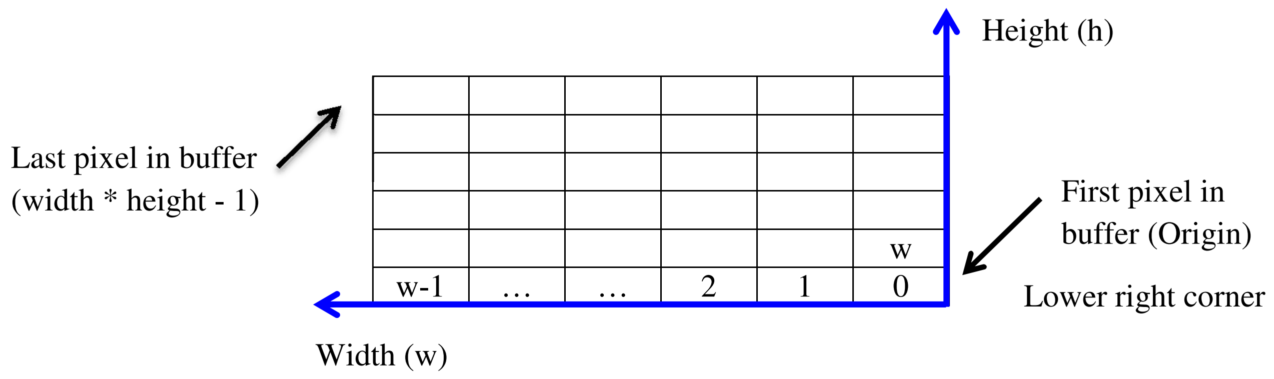
E-Puck lab #3 - Camera

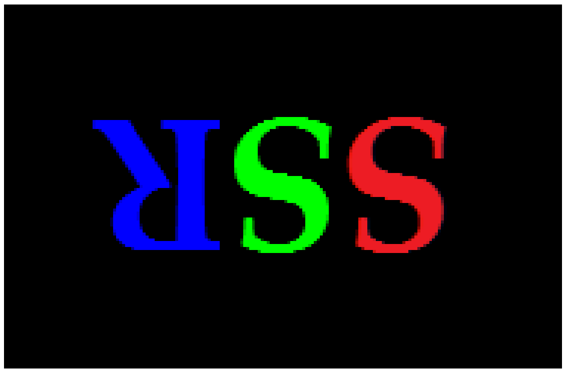
Now you know how to control the robot and how to ensure that it can avoid obstacles, it is therefore time to start using the camera. This will enable you to perform more interesting tasks such as object tracking and shape recognition.
In this lab you will initialize the onboard camera and read out single color pixels and rows of color pixels. The camera can also be used in grayscale mode, but using colors often makes you able to limit or omit shape recognition and instead rely entirely on colors. This is much faster, both for the processor to handle and, equally important, to develop.
The e-puck contains an onboard 640(h)x480(w) color camera. However the dsPIC processor running the epuck only has 8k of RAM which is not sufficient to store even a single image. This is one of the typical limitations of an embedded system. To be able to capture and process images in a sensible manner it is therefore necessary to control the picture resolution and the frame rate. A typical setup is a 1x80 pixel color image at 4 frames per second.
After capturing a picture a pointer will point to the buffer containing the picture data. The first pixel captured is the one located in the lower right corner seen from the perspective of the epuck. In a physical manner the picture can therefore be considered to be upside down.

Correlation between camera buffer indexing and physical orientation of camera



Below is an example of a multiline readout of 40 x 20 pixels.

What the camera is facing

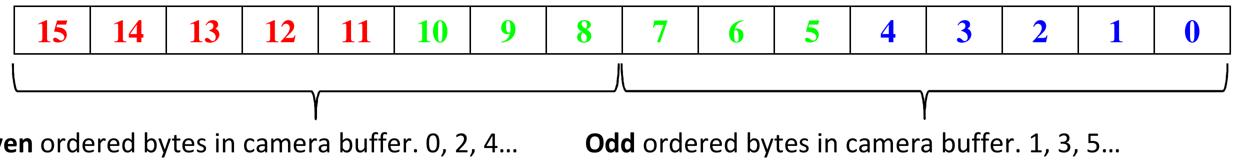


What the camera sees

```
........................................
........................................
........................................
.......RRRR.............................
.....RRRRRRRR....GGGGGGG.BBBBBBBBBB.....
....RRR...RRR..GGG....GG..BBBBBBBBBB....
...RRRR....RR..GG.....G...BBBB..BBBB....
...RRRR........GGG........BBBB..BBBB....
...RRRRRR......GGGG.......BBBB..BBBB....
....RRRRRRRR...GGGGGGGG...BBBB.BBBB.....
....RRRRRRRRR...GGGGGGGG..BBBBBBBB......
........RRRRRR.....GGGGG..BBBBBBBB......
..........RRRR.......GGG..BBBBBBBBB.....
...RR.....RRR..G.....GGG..BBBB.BBBB.....
...RR.....RRR..G.....GG...BBBB..BBBB....
...RRRRRRRRR..GGGG..GG...BBBBB..BBBBB...
....RRRRRR......GGGG..............BBB...
........................................
........................................
```

Camera data encoding

Different color modes exist. The most common is the RGB565 in which the colors and brightness's are organized in 2 bytes for each pixel as illustrated in the figure below. You should use that.


15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
Even ordered bytes in camera buffer. 0, 2, 4…
Odd ordered bytes in camera buffer. 1, 3, 5…

To search for an object of a specific color bit masking is therefore required.

Why is it called RGB565?

Make sure to use a contrast box with a uniform color while developing this code.
The first code you should implement is to setup the camera, read a single pixel of your choice, separate the colors by masking and then transmit those three bytes via Bluetooth to the terminal.
Use e_set_led(6, 2); to toggle one of the LEDs every time the camera function loops. This will let you know if it is still running.

Do you expect the values you get to be the exact same for other pixels? Why?
Check a couple of other pixels to check that the camera is working as you expect and confirm that the pixel values don't differ too much.

Which line do you need to select for the camera to capture data in a plane parallel to the ground? Test your prediction.

Next you should develop an algorithm that reads a line of 1 x 80 pixels in the plane parallel to the ground and searches for the area containing the highest concentration of red. The algorithm should then command the robot to turn towards this concentration, placing the robot facing straight towards the red object.