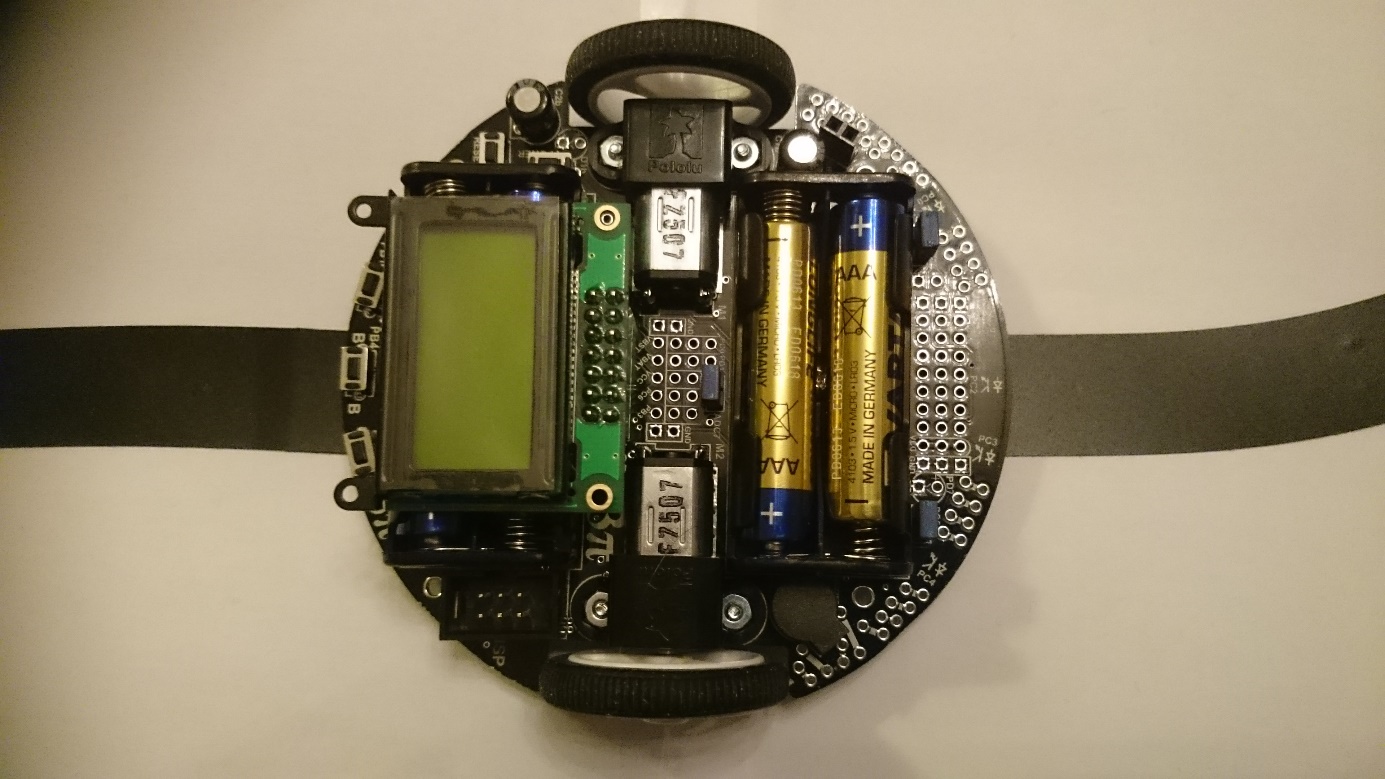
# The sample code

The project containing the sample code makes sure that the robot starts up properly, initializes all sensors etc. As part of this sequence it also calibrates the line sensors (too know how black and white looks). For this calibration to work you need to put the robot on a track, over the black line, in the way shown on the picture. The press button A, wait until it stops, press button A again, and after that it is your code that is running.

Pressing button A twice like this always needs to be done, but it is only in exercises that actually use the sensors that you need to care about placing the robot like in the picture.



Finally, and most importantly, this is where you put your code:

In pink\_programming.c, in this section:

// -----------------------------------------------------------------

// -----------------------------------------------------------------

void pink\_programming(void){

while(1){

// Pink programming: Put your code here

}

}

// -----------------------------------------------------------------

// -----------------------------------------------------------------

# API (Application programming interface)

## Motor functions

### Left motor

void set\_m1\_speed (sint16 speed);

### Right motor

void set\_m2\_speed (sint16 speed);

### Parameters

speed: between -100 and +100

### Example usage

**set\_m1\_speed (50);**

## LCD functions

### Clearing the display

void clear(void);

### Move print cursor on the LCD

void lcd\_goto\_xy(int col, int row);

### Print a text on the LCD

void print(const char \*str);

### Print a number on the LCD

void print\_long(long value);

### Another function for printing (more complicated but more flexible)

int printf(const char \*format, ...);

### Example usage

**clear();**

**lcd\_goto\_xy(0,0);**

**print(“Test”);**

**lcd\_goto\_xy(0,1);**

**print\_long(-25);**

## Reading line sensors

unsigned int read\_line(unsigned int \*sensorValues, unsigned char readMode)

### Parameters

sensorValues: The values read from each sensor will be stored here

The individual values can be anywhere between 0 and 2000, where 0 means entirely white and 2000 means entirely black.

readMode: Always set it to IR\_EMITTERS\_ON

### Return value

A value between 0 and 4000, estimating how the robot is placed with respect to the track. 2000 means that the robot is on the middle of the line (sensor PC2 is directly over the line). 0 means that the line is under sensor PC0, and 4000 means that it is under sensor PC4.

### Example usage

**unsigned int l;**

**unsigned int values[5];**

**l = read\_line(values,IR\_EMITTERS\_ON);**

## Delaying program execution

void delay\_ms(int delay);

### Parameter

delay in milliseconds

### Example usage

**delay\_ms(200);**

## Playing a sound

void play\_from\_program\_space(const char\* sequence)

### Example usage

**play\_from\_program\_space(beep); // (beep is defined in the pink\_programming.c file)**

## Reading button status

unsigned char button\_is\_pressed(unsigned char buttons)

### Parameter

Buttons: Can be one or a combination of BUTTON\_A, BUTTON\_B, BUTTON\_C or ANY\_BUTTON.

(one combination can for example be BUTTON\_A | BUTTON\_B )

### Example usage

*Wait for any button to be pressed*:

**while(!button\_is\_pressed(ANY\_BUTTON))**{

} // the return value will be non-zero if any of the buttons is pressed

*Check if the left button is pressed*

unsigned char button;

button = **button\_is\_pressed(**BUTTON\_A**)**

// If button A is pressed the value returned will be BUTTON\_A (non-zero), otherwise it will be 0

if (button)

{

// Do something

}

## Reading the colour sensor

colorType DetectColor (void)

typedef enum {

RED,

GREEN,

BLUE,

UNKNOWN,

} colorType;

### Example usage

**colorType myColor = DetectColor();**

**if(myColor == RED)**

**{**

**// Put code to pop the balloon!**

**}**

## Stopping the robot

void stop(void);

This function (implemented in pink\_programming.c file) sets the speed of both motors to 0 to stop the robot.

### Example usage

stop();

## For more functions or details (the advanced programmer)

See avr\_library\_commands.pdf on the desktop of the Virtual machine

# A number of tips

## In Geany

### Finding out more about a variable, constant or function

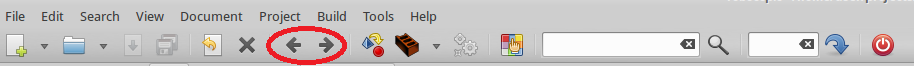
Select it and…

Right-Click -> Go to definition (or Press Ctrl+T)

Right-Click -> Find usage (Results are shown in the Messages tab on the bottom of the screen)

### Quicker navigation

The back and forward buttons are your friends!



# Exercises

## General

Hints for each exercise are available in the Hints.doc and full example code is available in Code.doc. However these only covers the basic exercises, for the rest you need to write code your own code (with our support of course!).

In addition, we have tape and paper, so make your own track or labyrinth if you want to!

## Basis exercises

1. Run the robot forward 50 cm, stop, turn 180 degrees, run forward 50 cm, go backwards 50 cm, stop
2. Add a sound before turning
3. Play the sound every 5th second instead
4. Print the text “Hello!” on first row, and “I am 3Pi” on the second row
5. Wait until someone presses the middle button, then run in a circle of 50 cm diameter
6. If someone instead presses the right button, run like a snake (run in a half circle to the right, then left, right again and so on, ca 20 cm diameter of the half circles).
7. Print the value of the line sensors (returned value, i.e. the estimated angle) on the first line. On the second line, print *right* if the value is greater than 2000 and *left* otherwise.

When finished, test to put the robot on the line on the track. See how the value changes depending on how you turn the robot.

Note: Now the robot needs to be calibrated properly, according to what is written in “The Sample Code“ chapter.

## Further exercises

1. Go forward until the robot finds a line, beep, then turn 180 degrees and continue
2. Try to stay inside a square of black tape by “bouncing” when finding the line
3. Find your way into the end of the labyrinth and play a sound when reaching there
4. While the robot is driving in the labyrinth, print the number of turns encountered so far on the display.

## Line following on the track

1. Make the robot follow the black line on the track
2. Try to go round the track as fast as possible (for example speed up on the straights and slowdown in the tight corners)

## Robocop mission

1. Find a marker on the track (you can optionally get ready made code for this step)
2. Pop a balloon (high speed is important!)
3. Detect the colour of a balloon, and pop it if it is red.
4. Try out the full Robocop mission (go around the track, find marker, pop balloon if red).