

Janno Jõgeva, Heilo Altin, Siim Sundla (Tartu Ülikool), 2013



## E-kursuse **"ROBOTICS"** materjalid

Aine maht 6 EAP

**Janno Jõgeva, Heilo Altin, Siim Sundla (Tartu Ülikool), 2013**

# Table of contents

## [Week 1 - Intro](#)

### [Initial setup](#)

#### [Installing software](#)

### [Practical assignment](#)

#### [The Aim](#)

#### [You will learn to](#)

#### [Tools used](#)

#### [Work description](#)

### [Homework Assignment](#)

#### [Description](#)

#### [Syntax](#)

#### [Sample execution](#)

#### [Input](#)

#### [Output](#)

#### [Sample images](#)

#### [Deliverable](#)

#### [Aim](#)

## [Week 2 - Video](#)

### [Practical assignment](#)

#### [The Aim](#)

#### [You will learn to](#)

#### [Tools needed](#)

[Work description](#)

[Homework Assignment](#)

[Description](#)

[Syntax for program execution](#)

[Sample execution](#)

[Deliverable](#)

[Aim](#)

[Week 3 - Video](#)

[Practical assignment](#)

[The Aim](#)

[You will learn to](#)

[Tools needed](#)

[Work description](#)

[Homework Assignment](#)

[Description](#)

[Deliverable](#)

[Sample program execution](#)

[Execution explanation](#)

[Week 4 - Obstacle avoidance - motion planning I](#)

[Practical assignment](#)

[Homework Assignment](#)

[Description](#)

[Deliverable](#)

[Sample program execution](#)

[Execution explanation](#)

[Test images](#)

## [Week 5 - Obstacle avoidance - motion planning II](#)

[Practical assignment](#)

[Homework Assignment](#)

[Description](#)

[Deliverable](#)

[Sample program execution](#)

[Execution explanation](#)

[Test images](#)

## [Week 6 - Obstacle avoidance - debug and optimization](#)

[Practical assignment](#)

[Homework Assignment](#)

[Description](#)

[Deliverable](#)

[Sample program execution](#)

[Test images](#)

## [Week 7 - Introduction to ROBOTC](#)

[Practical assignment](#)

[The Aim](#)

[You will learn to](#)

[Tools needed](#)

[Work description](#)

[Homework Assignment](#)

## [Week 8 - Positioning with 1st level robot](#)

[Practical assignment](#)

[The Aim](#)

[You will learn to](#)

[Tools needed](#)

[Work description](#)

[Homework Assignment](#)

[Aim](#)

[Description](#)

[Deliverable](#)

[Week 9 - Positioning with 1st level robot](#)

[Practical assignment](#)

[The Aim](#)

[You will learn to](#)

[Tools needed](#)

[Work description](#)

[Homework Assignment](#)

[Help](#)

[Description](#)

[Deliverable](#)

[Week 10 - Line following using PID](#)

[Practical assignment](#)

[The Aim](#)

[You will learn to](#)

[Tools needed](#)

[Work description](#)

[Hints](#)

## [Homework Assignment](#)

[Help](#)

[Description](#)

[Deliverable](#)

## [Week 11 - NAO - visual programming](#)

### [Getting started](#)

[Download software](#)

[NAOqi](#)

[Visual programming: Choregraphe](#)

[Simulation: Webots](#)

[Python programming](#)

[Connecting to a real robot](#)

### [Practical assignment](#)

[The Aim](#)

[You will learn to](#)

[Tools needed](#)

[Work description](#)

## [Homework Assignment](#)

[Description](#)

[Deliverable](#)

## [Week 12 - NAO - Python scripts](#)

### [Practical assignment](#)

[The Aim](#)

[You will learn to](#)

[Tools needed](#)

[Work description](#)

[Homework Assignment](#)

[Description](#)

[Deliverable](#)

[Hint](#)

[Evaluation](#)

[Week 13 - NAO - football exercise final](#)

[Practical assignment](#)

[The Aim](#)

[You will learn to](#)

[Tools needed](#)

[Work description](#)

[Homework Assignment](#)

[Description](#)

[Deliverable](#)

[Evaluation](#)

[Week 14 - NAO - final week](#)

[Appendix I - Top tips for solving issues](#)

[Camera synchronization](#)

[Bluetooth connection issues](#)

# Week 1 - Intro

## Initial setup

### Installing software

The following software is required to control LEGO NXT using an external computer with a camera. Python 2.6 is used due to pyBluez. Libraries for robot and vision are listed in the order of dependencies. This setup is for Windows systems but can be adjusted to other operating systems.

### **Python 2.6, 32bit**

<http://www.python.org/download/releases/2.6.6/>

### **Libraries for controlling the robot (For practice sessions)**

pyBluez Bluetooth library for Python (only for Python 2.6)

<http://code.google.com/p/pybluez>

nxt-Python for robot interactions

<http://code.google.com/p/nxt-python>

### **Libraries for image processing**

Numpy – Scientific computing tools (1.6.1)

<http://sourceforge.net/projects/numpy/files/NumPy/1.6.1/numpy-1.6.1-win32-superpack-python2.6.exe/download>

SciPy – Scientific Tools for Python (0.9.0)



<http://sourceforge.net/projects/scipy/files/scipy/0.9.0/scipy-0.9.0-win32-superpack-python2.6.exe/download>

OpenCV computer vision library (2.4.2)

<http://sourceforge.net/projects/opencvlibrary/files/opencv-win/>

## **Environment variables**

To be able to use OpenCV with python we must modify environment variables.

On Windows 7 navigate to Control Panel ->System and Security ->System and click 'Advanced System Settings'. From the new opened window click 'Environment Variables'.

Add or modify user variables to include:

**PATH** – add python 2.6 binary directory, this enables binaries to run from a command line using binary name as a command

Like: C:\Python26

**PYTHONPATH** – add a location to OpenCV Python 2.6 libraries

Like: C:\Downloads\opencv\build\python\2.6

## **Git for Windows (msysGit)**

Installation of msysGit can be done with preselected options, however, no need to install the "Windows explorer integration". You can just uncheck it while installing Git.

<http://code.google.com/p/msysgit/downloads/list?can=2&q=%22Full+installer+for+official+Git+for+Windows%22>

## TortoiseGit 1.8.1

This gives you the right-click options seen in practice session.

<http://code.google.com/p/tortoisegit/wiki/Download?tm=2>

## Practical assignment

### The Aim

This lab aims at getting a first look at machine vision. It will also give an overview of the tools used in this course.

### You will learn to

1. Read, write and modify images using OpenCV
2. Generate [histograms](#) and do basic thresholding on single images.
3. Work with pixels. We will need this for our first homework assignment.

### Tools used

1. Python 2.6
2. OpenCV library

### Work description

#### **1. Learn about using OpenCV**

Consider the following sample:

```
import cv2

cv2.namedWindow("Sample")

img = cv2.imread("sample.jpg")
```

```
img[0][0] = [255,255,255]
cv2.imshow("Sample", img)
while True:
    ch = cv2.waitKey(5)
    # Kill program when ESC pressed
    if ch == 27:
        break
cv2.destroyAllWindows()
```

Get it to work. What do these commands do?

Read about OpenCV images and video from:

[http://docs.opencv.org/modules/highgui/doc/reading\\_and\\_writing\\_images\\_and\\_video.html](http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html)

OpenCV image is a three-dimensional matrix in BGR colour space (reversed RGB) of form **img[row][column][channel]**, indexes start from upper left corner of an image. For example, to access a value of a green channel pixel in the left lower part of 640x480 image saved in img variable we would write: `img[479][0][1]`

## 2. Create a thresholded image

Create a Python function that sets pixels white or black based on their colour. Display the result on screen

INPUT: image matrix to process, colour channel to process (B/G/R), threshold 0-255

OUTPUT: (none, changes can be made on same image)

For example call to `myThresholdFunction(img, 0, 100)` could mean that all pixels in image img that have blue component intensity below 100 should be coloured black, all others should be white

When you are ready show the result to your supervisor

### 3. Draw a histogram from input images.

Create a Python function that creates an image that represents the **histogram** of input image (for one colour channel or all). Implement **histogram** calculation in your own code, do not use OpenCV function for that. **Histogram** and the original image should be displayed on screen. NB! You need to create an empty image for your **histogram** result (3 dimensional array).

INPUT: image matrix to process

OUTPUT: image matrix representing **histogram**

1. Use OpenCV functions to read an image from a disk
2. Use your function to create **histogram** image
3. Display result on screen

You may find the following numpy functions useful (import numpy first):

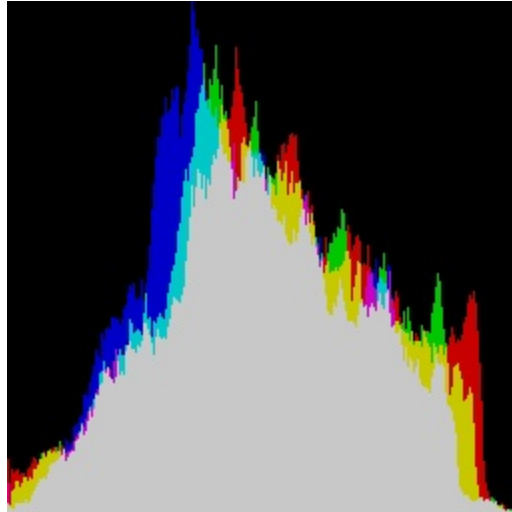
Create empty 256x256 image:

```
img = numpy.zeros((256,256,3), numpy.uint8)
```

Find max value from an array or matrix:

```
numpy.amax(array)
```

What your result may look like (all channels on top of each other version):



When you are ready show the result to your supervisor

## Homework Assignment

### Description

Before starting with the homework make sure you have all of the software set up and you have cloned your GIT repository. If you run into any problems you can ask for assistance in the "Help" forum.

Now download and extract the included .zip file. You should add the files so that your **hw01.py** file and the **hw01/** folder are in your root folder (**robotika-xIDx**).

This homework concentrates on finding an object from a set of images. Write a solution in Python 2.6. You can also use libraries listed under the software section ([OpenCV](#), Numpy, etc.).

Inside your program create a reusable function for detecting the "centre of mass" of of an area with a specific colour.

## Syntax

```
python hw01.py folder/input_image.tiff blue_value green_value  
red_value deviation
```

## Sample execution

```
> python hw01.py hw01/sample01.tiff 200 200 200 20
```

```
> [(100, 200), (245, 428)]
```

This means two areas were found with their centers at coordinates (100, 200) and (245, 428).

## Input

Image, colour to search for B,G and R components, allowed deviation for colour. This means that you have to find all areas that are in the allowed range.

Allowed deviation means that the value of every colour channel has to stay in the range. Let's again consider the sample used before and see if a pixel's colour is suitable for the area or not.

```
> python hw01.py hw01/sample01.tiff 200 200 200 20
```

```
[0, 0, 0] - not in range
```

```
[190, 57, 100] - not in range
```

```
[200, 200, 200] - in range
```

```
[180, 200, 220] - in range
```

```
[180, 200, 221] - not in range
```

## Output

A rounded arithmetic average of both x and y indexes of matched pixels (a centre of this colour). A list of tuples. Coordinates should be rounded before output so that instead `[(2.50, 4.49)]` you return `[(3, 4)]` where 3 is the number for the row and 4 for the column.

Once started your final programme should not wait for any user input and should not display any results other than regular print of the results. I do encourage you to display the results graphically when testing the programme.

## Sample images

There is a single white pixel in the `sample01.tiff` it is located at `(100, 200)` this should help you to debug your solution. These images are for you to test your code on.

## Deliverable

The solution should be in your git repository by the deadline. This means you should commit and push safely before the deadline. As the images take ~10MB it is a good idea to do the first push as soon as you start working. Only changes are pushed so the next time you will get a quick push. Do not forget to add all of the needed files before pushing.

## Aim

All of the solutions will be benchmarked to see how well they perform. So the aim is to detect all of the areas precisely and as quickly as possible.

# Week 2 - Video

## Practical assignment

### The Aim

This week we will look into one of the possible localization methods in robotics - a fixed overhead camera. By the end of this session we will have a programme that can track a colored area on the image.

### You will learn to

1. Connect a webcam to your program
2. Detect simple objects from video stream

### Tools needed

1. Python 2.6
2. OpenCV library (Numpy)
3. Logitech C920
4. Colored paper

### Work description

1. Clone your repository to the computer. If you already have it pull the latest version to avoid conflicts.
2. Add a new file `/robotika-yourID/pr02.py` for working in this practice session.
  1. NB! If you rename any of your files in the repository use the right click option "rename" from tortoises' menu. If you just rename it you will loose some of the consistency in between the versions. If you can not find the option from the menu ask for help.
  2. When you need to use a function from lets say your first homework assignment just add `import`



hw01 in your `pr02.py` file and you can use the functions as `hw01.your_function()`.

3. Connect your camera to a python programme.
  1. Connect the webcam provided by lab supervisor to your computer. You can use device manager to make sure that the camera was connected correctly. It should also be indicated in the lower right corner on your screen.
  2. Write a small function to display the video from camera. You can find documentation from [docs.opencv.org/modules/highgui/doc/reading\\_and\\_writing\\_images\\_and\\_video.html#videocapture](https://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html#videocapture)
4. Commit your code - this gives you a point to come back to when you need to remind yourself how it was done.
5. Now we will use our functions from last practice session on the camera image. What do we need to change in the code?
  1. First try to grab a single image and apply your function on it.
  2. Write a loop to do it for multiple images. You might want to add the possibility to wait for a key to grab the next image. You should have some useful code from the last practice session.
  3. Can your laptop handle the load?
6. Try to detect a piece of paper (a spot). Draw a rectangle on its centre.
  1. First let's use the function you completed at home to find a colored paper from an image.
  2. Print out the coordinates of the spot.
  3. Find a function from the [OpenCV documentation](#) for drawing a rectangle and use it.
  4. Try to find good color parameters to get a stable location for the paper.
7. Commit your code.
8. Choose another colour for the target. Draw a circle on it.
  1. Again let's use the same function to detect the destination.
  2. Use another piece of colored paper.

9. Move the spot on the field (manually) - can you still detect it?
  1. What problems may arise with detecting the location of the marker (paper)?
10. How do we get there? Draw a line on the image. From the spot to destination. (optional)
11. Commit and push your code and delete the repositories folder if it is a public computer.

## Homework Assignment

### Description

Before starting with the homework make sure you have all of the software set up and you have cloned your GIT repository. If you run into any problems you can ask for assistance in the "Help" forum.

You should add the files so that your hw02.py file and the hw02/ (create it in your repo) folder are in your root folder (robotika-xIDx). You can also try the functions on the images in hw01/ or take your own images if you have a webcam - feel free to push these images in the repository as well. But also keep in mind that adding very many images or large video files to the repo will make cloning considerably slower.

This homework concentrates on finding an object from a set of images. Write a solution in Python 2.6. You can also use libraries listed under the software section (OpenCV, Numpy, etc.).

Inside your program create a reusable function for detecting the "centre of mass" for the largest area with the specified colour. This means that your function returns the center of only the largest detected area.

## Syntax for program execution

```
python hw02.py folder/input_image.tiff blue_value green_value  
red_value blue_dev green_dev red_dev
```

## Sample execution

```
> python hw02.py hw02/sample01.tiff 200 200 200 20 10 30
```

```
> [(100, 200)]
```

This means that the largest area with the specified colour [200 200 200] that was detected has its' center at coordinates (100, 200).

Input: image, colour to search for B,G and R components, allowed deviation for colour. This means that you have to find all areas that are in the allowed range.

Allowed deviation means that the value of every colour channel has to stay in the range. Let's again consider the sample used before and see if a pixel's colour is suitable for the area or not.

```
> python hw02.py hw02/sample02.tiff 200 200 200 20 10 30
```

```
[0,0,0] - not in range
```

```
[190,57,100] - not in range
```

```
[200,200,200] - in range
```

```
[180,200,220] - in range
```

```
[180,200,231] - not in range
```

Output: a rounded arithmetic average of both x and y indexes of matched pixels (a centre of this colour). A list of tuples. Coordinates

should be rounded before output so that instead  $[(2.50, 4.49)]$  you return  $[(3, 4)]$  where 3 is the number for the row and 4 for the column.

Once started your final programme should not wait for any user input and should not display any results other than regular print of the results. I do encourage you to display the results graphically when testing the programme.

Sample images: there is a single white pixel in the `hw01/sample01.tiff` it is located at  $(100, 200)$  this should help you to debug your solution. These images are for you to test your code on.

### **Deliverable**

The solution should be in your git repository by the deadline. This means you should commit and push safely before the deadline. As the images take  $>10\text{MB}$  it is a good idea to do the first push as soon as you start working. Only changes are pushed so the next time you will get a quick push. Do not forget to add all of the needed files before pushing.

### **Aim**

All of the solutions will be benchmarked to see how well they perform. So the aim is to detect all of the areas precisely and as quickly as possible.

# Week 3 - Video

## Practical assignment

### The Aim

Get a first experience with control algorithms.

### You will learn to

1. Move robot by sending commands from software.
2. Moving by using simple feedback mechanism.

### Tools needed

1. Python 2.6
2. [OpenCV](#) library
3. nxt-python library
4. Webcam Logitech C920
5. LEGO robot

### Work description

1. Connecting the robot to your computer.
  1. [Turn](#) the robot on and activate its Bluetooth
  2. Make sure Bluetooth is activated and operational on your computer
  3. Go to Control Panel and navigate to "Hardware and Sound" -> "Devices and Printers" -> "Bluetooth devices".
  4. Click "Add a device". NXT should be visible on the list. Select it and enter pairing code on robot and computer
2. Moving the robot using Python.

Analyse and tryout the following example:

```
from nxt.motor import *  
  
import nxt  
  
b = nxt.locator.find_one_brick()  
  
l_motor = Motor(b, PORT_C)  
  
r_motor = Motor(b, PORT_B)  
  
motors = SynchronizedMotors(l_motor, r_motor, 1)  
  
motors.turn(100, 360)
```

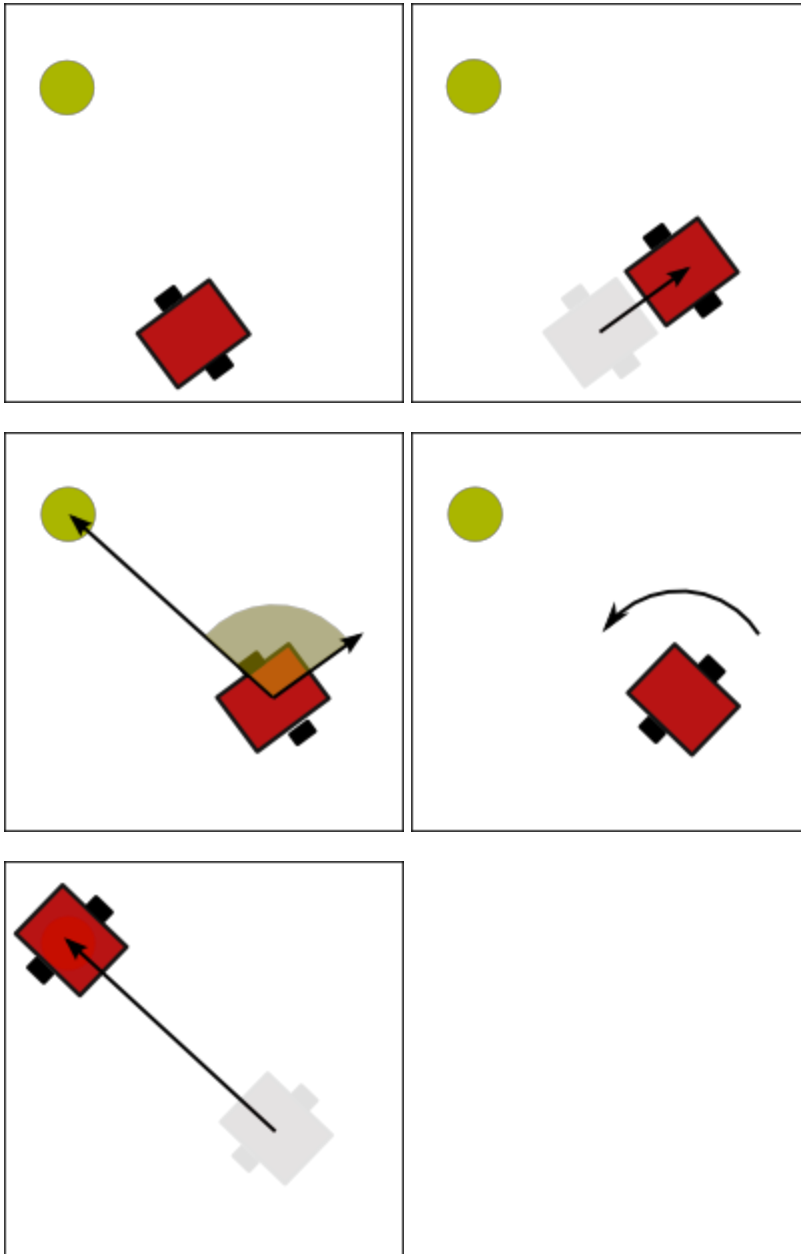
What does this code do? Figure out how to make the robot [turn](#). You can also find examples and commented source code from `nxt-python` installation directory.

### 3. Getting to the destination.

The goal of this exercise is to drive with a robot to a given destination. Both robot and destination have different colour markings. Locations and orientation is determined using an external camera.

The algorithm for achieving this task is the following (we will finish its implementation in next practice class):

1. Locate robot on an image
2. Drive forward
3. Locate robot again and calculate its orientation based on this and previous location
4. Locate destination and calculate its orientation from the robot
5. Calculate the difference between two orientations
6. [Turn](#) robot by the angle from the last step
7. Drive forward
8. Repeat from step 3 until robot has reached its destination



Or come up with a better way of doing the same thing.

Your task for this class is the following:

1. Write a function that calculates a distance between two points (in pixels)
2. Write a function that turns the robot by a given angle (preferably radians)

3. Try to get the presented navigation algorithm working using manual input (instead of measuring angles and distances from an image)
4. Commit and push your code and delete the repositories folder if it is a public computer.

## Homework Assignment

### Description

This time homework concentrates on finding the heading and the desired direction. We will try to calculate the moving direction of the robot. In addition we will write a function to calculate the angle between our moving direction and desired moving direction.

You advised but do not have to use the following structure for your solution:

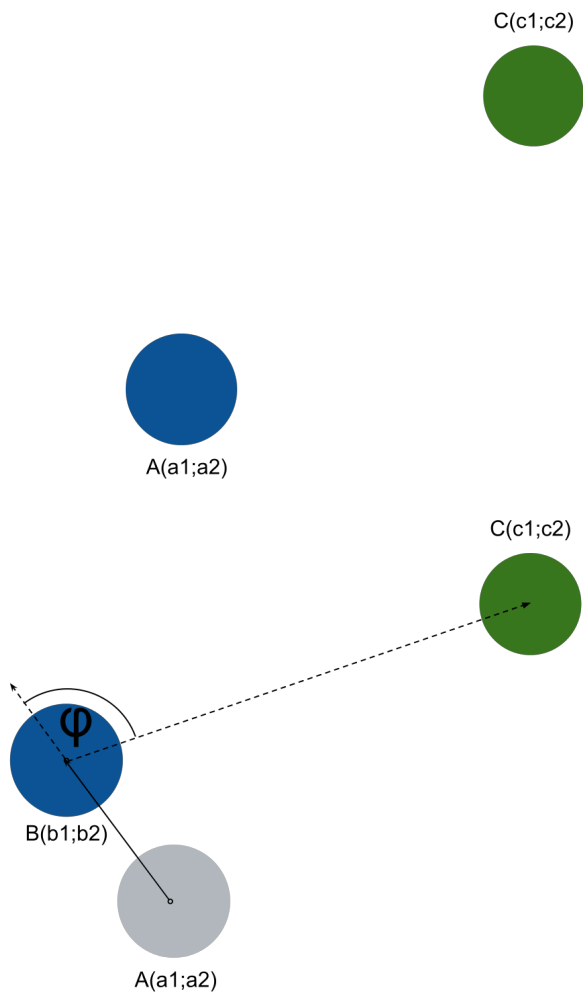
#### **1. Write a function that calculates robots orientation from the information you get from the two images.**

Remember that you can consider the image as a 2D graph and use the pixel coordinates as regular coordinates. So you can just use the line from point A to point B.

#### **2. Write a function that calculates a difference between two orientations**

The previous function enabled us to calculate the direction of a movement, this one can be used to tell how much the orientation must be changed to reach point C.





## Deliverable

Your solution should be contained in a file named `robotika-yourID/hw03.py`. You can and should use functions from previous homework assignments. You will probably need to use functions from the [python math](#) library.

## Sample program execution

```
> python hw03.py hw03/sample01.tiff hw03/sample02.tiff 200 40 50
20 10 30 100 80 200 20 10 30

> [(100, 200), (150, 250), (350, 50), 90]
```

## Execution explanation

```
> python hw03.py first_frame second_frame robot_BGR robot_BGR_dev
destination_BGR destination_BGR_dev

> [(a1,a2), (b1,b2), (c1,c2),  $\phi$ ]
```

- robot\_BGR - Robot marker colours (Points A and B in the sample above).
- robot\_BGR\_dev - Allowed deviation of robots marker colour.
- destination\_BGR - Robot marker colours (Points C in the sample above).
- destination\_BGR\_dev - Allowed deviation of robots marker colour.
- $\phi$  - angle between your moving direction and desired direction. For the desired heading use points B and C. Its value has to be between 180 and -180. No decimal places should be given in the output. Positive angles indicate left turn negative angles right [turn](#).

## Week 4 - Obstacle avoidance - motion planning I

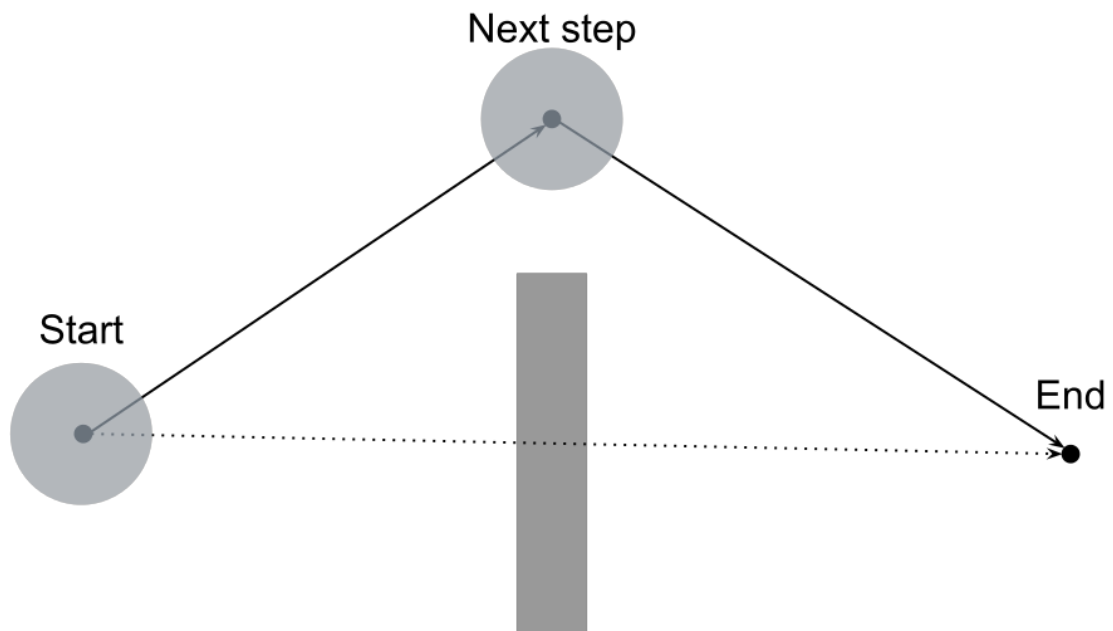
### Practical assignment

We will have time to complete the last weeks assignment. It contained many new concepts so it is reasonable to work on them a bit more to make sure that you get deep enough understand of these terms.

### Homework Assignment

#### Description

In this homework we look at the task of finding a usable path to the finish. What problems may arise?



If there wasn't for the wall on the way the robot could simply set its next destination to the end point and drive there. Now we have to take more steps to arrive at the end. To keep our solution as general as possible we only output the next step to be taken by the robot.

## Deliverable

Create a program that finds a usable path in the given configuration space and outputs only the next step. Your program should return the coordinates of the end point after doing a finite number of steps. Your solution for the homework should be in file named hw04.py. Feel free to have a separate file for functions and just read the arguments and call the function in the hw04.py file. Your solution should not return anything else than just the coordinates for the next step.

Do not forget to push your solution on time!

## Sample program execution

```
> python hw04.py hw04/sample01.tiff 100 200 500 600 20  
> [(300, 200)]
```

## Execution explanation

```
> python hw04.py sample_configuration start_rc end_rc robot_radius  
> [(next_row, next_column)]
```

- sample\_configuration - a threshold image that contains the areas that you can not enter.
- start\_rc - the location (row, column) your robot starts at.

- `end_rc` - the location (row, column) your robot wants to arrive to. If there are no obstacles you can just return this value. If there are obstacles on the direct route you have to return the next step your robot would take to get to the end location.
- `robot_radius` - This is the area that your robot takes in this configuration.
- `next_row` - Row coordinate to where you would move your robot in the next move.
- `next_column` - Column coordinate to where you would move your robot in the next move.

### Test images

At the end of this assignment you can find a set of test configurations. These images represent the frame you got from your camera. It is in black and white because you do not need to do obstacle detection in this homework assignment. Your program will have to find a way from start to end by not using any of the white pixels. On the `sample01.tiff` your program should return the endpoint as it next step as there are no obstacles.

# Week 5 - Obstacle avoidance - motion planning II

## Practical assignment

Working on enhancing performance of previous solution. As we will need to use our previous solutions for next tasks we will need to optimise our code to make it fast enough.

## Homework Assignment

### Description

This weeks assignment helps us to connect image plain with real world coordinates. By completing this task you should be able to tell how many meters to pixels in the image are apart in the real world. Now think back to the points A and B from HW03. What could be the use for such function? We will also make use of this functionality in the sensor based navigation part of the course.

You can assume that the optical axis of the cameras lens is collinear to the normal of the floor that holds the paper strip. You can also consider the camera as a theoretical pinhole-camera - without any distortion.

Keep in mind that you should already have most of the code you need for this assignment from the previous ones!

### Deliverable

You should have a file named hw05.py in your repository by the deadline. Feel free to have a separate file for functions and just read

the arguments and call the function in the hw05.py file. Your solution should not return anything but the asked output.

Do not forget to push your solution on time!

### Sample program execution

```
> python hw05.py hw05/sample01.tiff 20 20 200 15 15 50 300
```

```
> 500
```

### Execution explanation

```
> python hw05.py frame strip_BGR strip_BGR_div len_strip
```

```
> pixels_per_meter
```

- frame - Image to find the calibration strip from
- strip\_BGR - Marker colours
- strip\_BGR\_div - Accepted difference from the marker colours
- len\_strip - Length of the longer side of the calibration strip in mm. You can assume that the other side is at least 3 times smaller. I.e. if the parameter is set to 300 mm you can assume that the other side is no longer than 100 mm
- pixels\_per\_meter - This should be a positive integer that represent the number of pixels that represent one meter on the image. Lets say that you output 500 and you are trying to track your robot. First you find the robot at location (50,50) and on the second frame from (100,50) given your previous output  $50/500=0,1$ . This means that the robot has moved about 0,1 meters or 10 cm between the two frames.

## Test images

On the first test image there is a strip with dimensions 100x10 pixels. Lets say that the real world length of the longer side of the paper strip is 20 cm. When executing your program with arguments:

```
> python hw05.py hw05/sample01.tiff 255 255 255 1 1 1 200
```

it should return:

```
> 500
```

On the other images the actual length of the longer side of the paper strip is 297 mm. But you can use any value for testing.



# Week 6 - Obstacle avoidance - debug and optimization

## Practical assignment

As you have probably noticed by now correct colours are very important to make our solution reliable enough.

## Homework Assignment

### Description

This week weeks homework guides us to next part of the course. We will try to write a program that could be used to track our robots in the upcoming practice sessions. The idea is to get something similar to a GPS device. You will most likely need `cv2.putText()`. Keep in mind that to use predefined variables like "FONT\_HERSHEY\_SIMPLEX" you need to append the library name: `cv2.FONT_HERSHEY_SIMPLEX`.

### Deliverable

You should have a file named `hw06.py` in your repository by the deadline. Feel free to have a separate file for functions and just read the arguments and call the function in the `hw06.py` file. This time you have to open a video device to display the output! On the image you should put text that gives the following list of properties for every frame:

1. Frame ID - An incrementing number starting from 0.
2. Location - The coordinates for the coloured marker (i.e. robot) in pixels.

3. Speed - The speed of the robot in pixels/s.
4. Average speed - The average speed of the robot in pixels/s.  
You have to take the average from at least 10 frames.
5. FPS - Frames per second that your solution is achieving.

You are allowed to add more features to your solution.

Do not forget to push your solution on time!

### Sample program execution

```
> python hw06.py 10 20 200 10 20 50 -1
```

### Execution explanation

```
> python hw06.py robot_BGR robot_BGR_div video_dev_id
```

- robot\_BGR - Robot marker colours (Points A and B in the sample above).
- robot\_BGR\_dev - Allowed deviation of robots marker colour.
- video\_dev\_id - ID for the video device.

### Test images

Use of your own webcams is encouraged. If you do not have any webcam to use just use a random video file for testing. If you run into problems post in the forum.

# Week 7 - Introduction to ROBOTC

## Practical assignment

### The Aim

Learn how to use ROBOTC, MINDSTORMS sensors, NXT motors movement, synchronisation.

### You will learn to

1. Use distance sensor(sonar) to detect distances and make the robot act on that
2. Movement details of MINDSTORMS robot, use of synchronization

### Tools needed

1. ROBOTC 3.x or newer
2. USB cable
3. Tape measure
4. LEGO robot

### Work description

1. First we are going to learn how to move the robot

ROBOTC is a useful tool for programming following platforms:

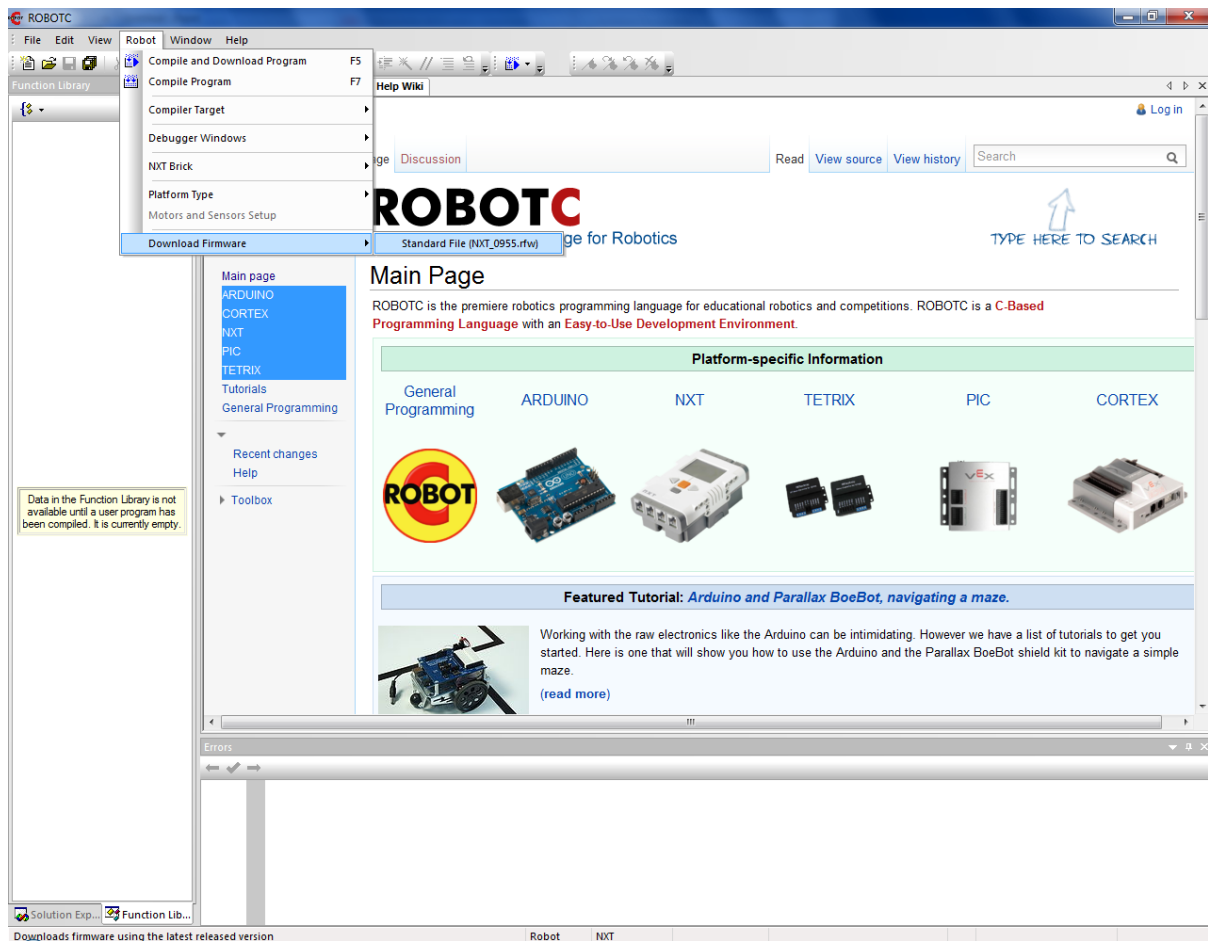
- ARDUINO
- CORTEX
- NXT

- PIC
- TETRIX

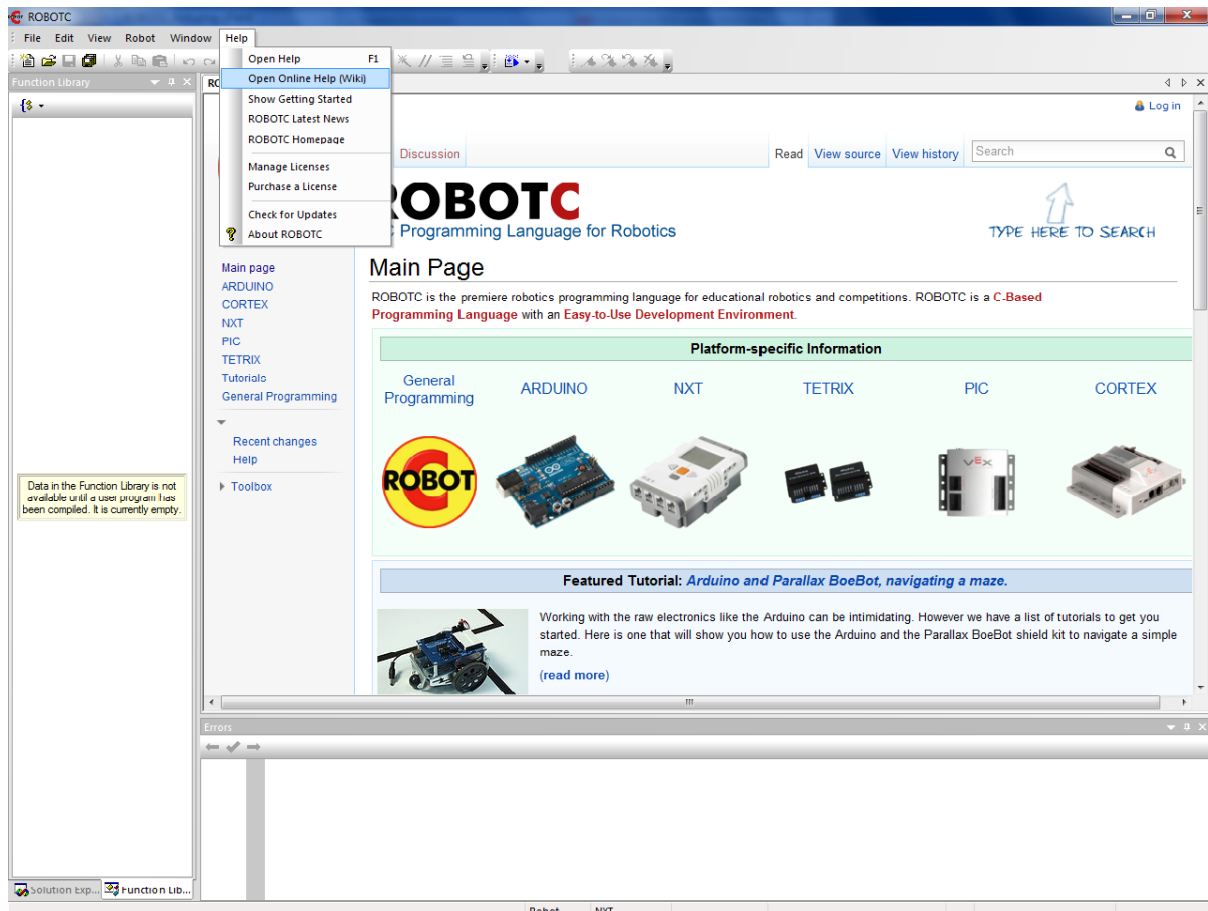
Main difference between programming NXT in Python and ROBOTC is that ROBOTC compiles and downloads the program to the robot so computer has no control over the robot. Programs are executed in the robot, not in the computer.

1.1 Open the ROBOTC program from Desktop.

1.2 Connect the NXT with USB cable.



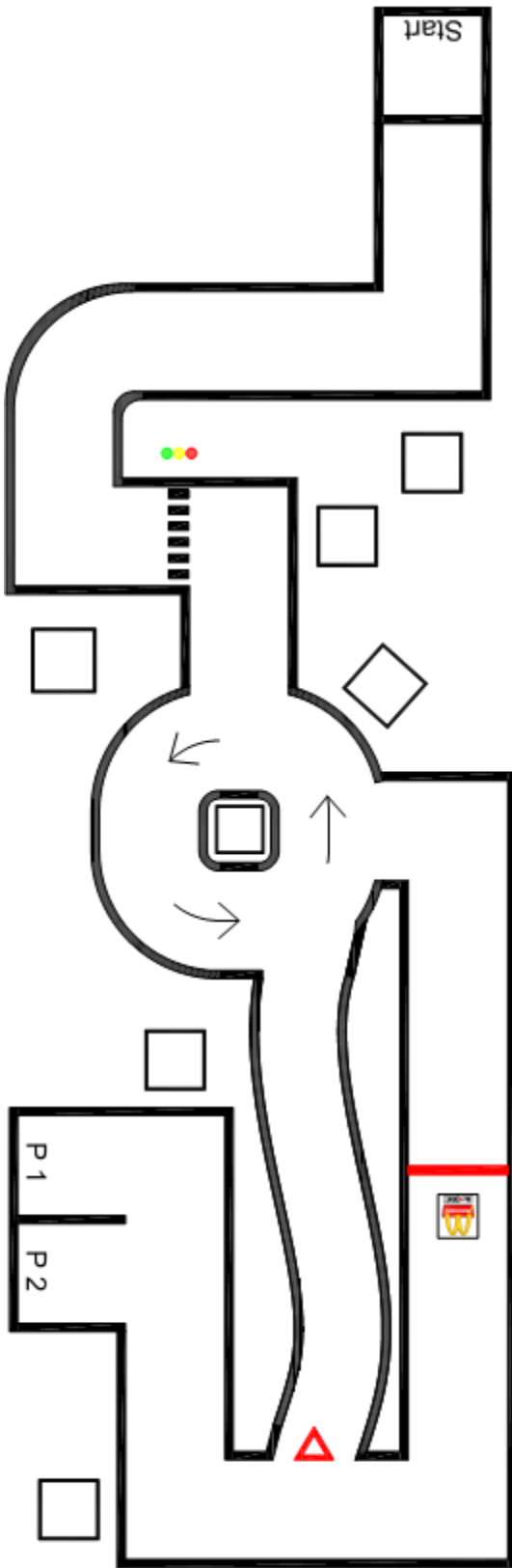
1.3 Download the ROBOTC firmware to the robot (Tools->Download Firmware)



1.4 You can use ROBOTC online help Wiki. There is all the information you need to know about programming NXT in ROBOTC.

2. Make the robot to go straight 200 cm.

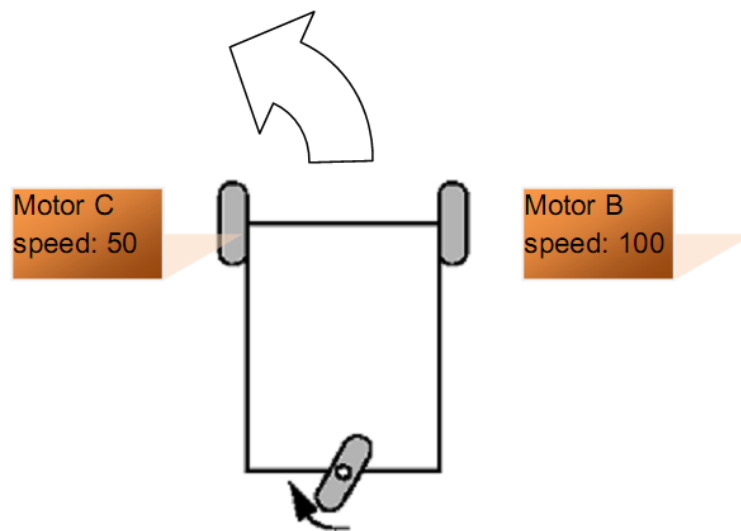
3. Make the robot to follow the pathline in a small city. Start from "start" square and finish in one of the parking lots.



**nSyncedMotors = synchBC** means that motor C is a slave to motor B. Motor C can follow motor B speed by 100 % but cannot exceed that!

**nSyncedTurnRatio=100** means that motor C is following motor B speed 100 %

**nSyncedTurnRatio=50** means that motor C is following motor B speed 50 % and is causing the robot to turn left.



## Homework Assignment

This week we will have a test in moodle environment.

# Week 8 - Positioning with 1st level robot

## Practical assignment

### The Aim

Learn how to use open loop control to move the robot to the position  $(x,y)$  and check it's coordinates.

### You will learn to

1. Drive to position  $(x,y)$  when robot initial position is  $(0,0)$  with no obstacles
2. Make the robot to calculate it's angle
3. Make the robot to calculate it's coordinates.

### Tools needed

1. ROBOTC 3.x or newer
2. USB cable
3. Tape measure
4. LEGO robot

### Work description

1. Move the robot to position  $x=25$  cm,  $y=25$  cm and see if robot is showing right coordinates. Robot moving direction is facing x axis in initial position.

1. 1 Make the robot to calculate angle between it's own direction and x axis. Let the robot to show angle on screen and [run](#) some tests like this:

```
nMotorEncoder[motorB]=0;//zero encoder
```



```

nMotorEncoder[motorC]=0;//zero encoder

nSyncedMotors = synchBC;

nSyncedTurnRatio = 0;

nMotorEncoderTarget[motorB] = 190;

motor[motorB]=50;

while(nMotorRunState[motorB] != runStateIdle) { //while motor B is
still running

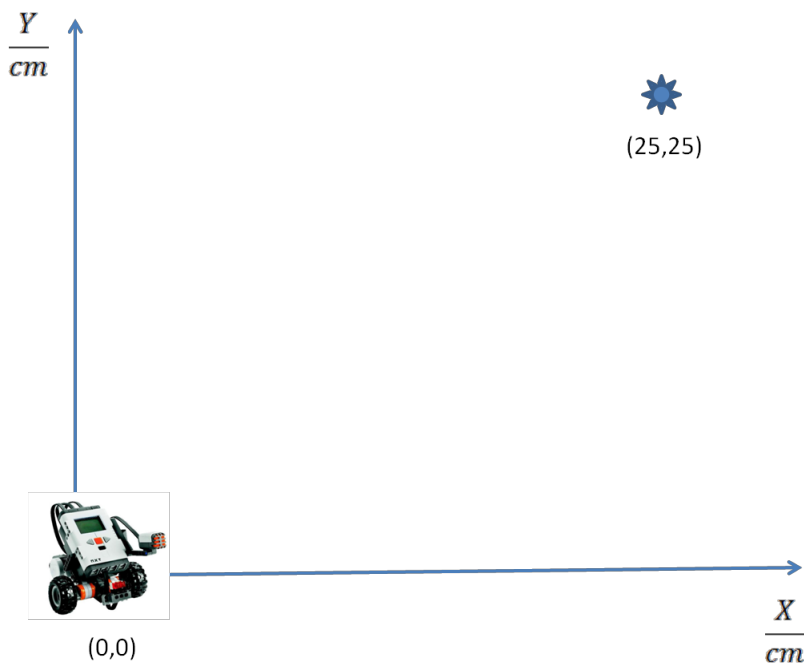
//do here what needed

} //while

motor[motorB]=0;

```

This would make the robot in standard Mindstorms configuration to turn 45 ° left.



2. Start calculating x, y coordinates. Robot should show it's final position for 5 sec on screen as shown here:

X: 25 cm

Y: 25 cm

Run some simple tests at the beginning. Have robot **turn** first and then drive forward. Later have the robot to drive curves and find it's location.

When using `nMotorEncoderTarget[motorX]`, remember that it is not cumulative.

For example:

When you want motor B to **turn** 90 degrees:

```
nMotorEncoderTarget[motorB]=90;
```

```
/---Motor B turns---//
```

and you want motor B to **turn** again 90 degrees:

```
nMotorEncoderTarget[motorB]=90;
```

## Homework Assignment

### Aim

This makes you think over what you learned and prepare you for the next week. Theoretical information will be available from book "[Intro to Autonomous Mobile Robots](#)". You can also ask from forum.

You can install [ROBOTC 30 day trial](#) to your computer with "[Virtual World](#)" which enables you to test your function in your computer. You have to open from "Robot" menu "Global variables".

## Description

This homework gives you further experience in calculating robot's position in x,y plot and  $\alpha$  and  $\Delta x$  and  $\Delta y$ .

## Deliverable

Last practise session you programmed robot that calculated it's x and y position as a function. Your homework is to complete that function and calculate also angle  $\alpha$  and  $\Delta x$  and  $\Delta y$ , see Fig below.  $\alpha$  is the angle between the robot direction and target direction. Alpha can have positive value or negative value when robot has turned more other way.  $\Delta x$  and  $\Delta y$  are difference between robot's current position and destination coordinates.

This program has to work as a function like following:

```
float x_dest=25;//Destination on x axis in cm

float y_dest=25;//Destination on y axis in cm

task position(Motor_B_degrees, Motor_C_degrees){

nxtDisplayStringAt(0, 60," $\alpha$ : ");

nxtDisplayStringAt(30, 60, "%.1f",calc_alpha); //Here you print
calculated alpha on screen

nxtDisplayStringAt(0, 40, " $\Delta x$ : ");

nxtDisplayStringAt(30, 40, "%.1f",calc_ $\Delta x$ ); //Here you print
calculated  $\Delta x$  on screen

nxtDisplayStringAt(0, 20, " $\Delta y$ : ");

nxtDisplayStringAt(30, 20, "%.1f",calc_ $\Delta y$ );//Here you print
calculated  $\Delta y$  on screen
```

```
wait1Msec(5000);
```

```
}//task
```

Your function will be controlled on a robot with given motor B and motor C rotation values.

For example:

```
position(5,3);// it means that Motor B has turned 5 degrees and  
Motor C has turned 3 degrees. Now you have to calculate x, y  
coordinates,  $\Delta x$ ,  $\Delta y$  and  $\alpha$ .
```

# Week 9 - Positioning with 1st level robot

## Practical assignment

### The Aim

Learn how to use feedback control to move the robot to the position  $(x,y)$ .

### You will learn to

1. Drive to position  $(x,y)$  when robot initial position is  $(0,0)$  with no obstacles and calculating drive speed  $v$  and rotational speed  $\omega$ .

### Tools needed

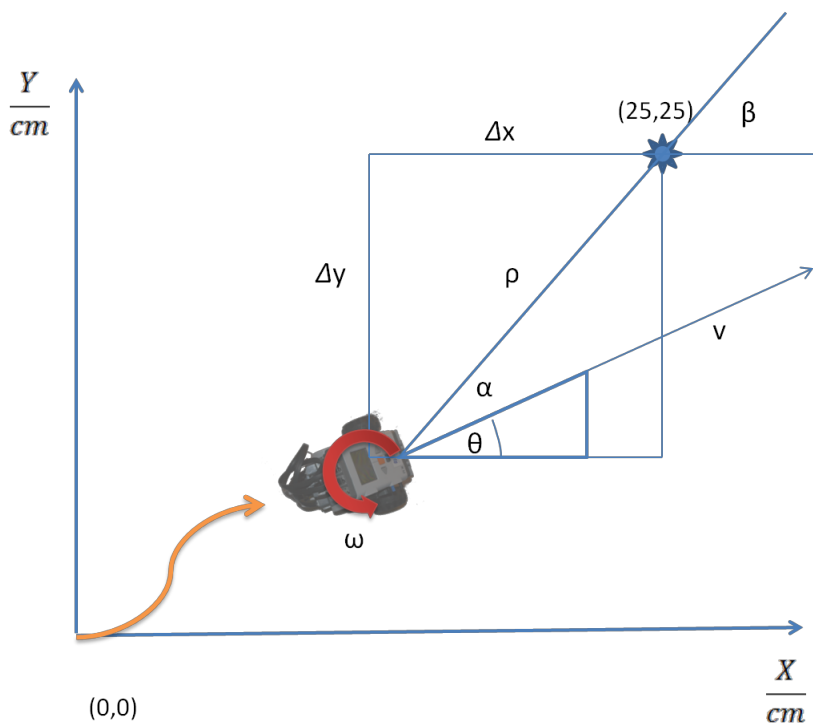
1. ROBOTC 3.x or newer
2. USB cable
3. Tape measure
4. LEGO robot

### Work description

1. Make the robot to calculate it's current location. Use the algorithm seen in seminar.
2. Set destination point as  $(25,25)$ .
3. Make the robot to go that point statically. Let the robot to calculate it's position and check if you code works.
4. Start calculating:

1.  $\Delta x, \Delta y$  - should be changing towards zero
2.  $\alpha, \beta$  - alpha should be driving towards zero, beta should be driving same as  $\theta$
3.  $\rho$  - should be driving towards zero
4.  $v$  - should be driving towards zero

5. Make the robot go to position as a function of  $v$  and  $\omega$ . There are many possibilities to make the robot to follow a line.



## Homework Assignment

## Help

This makes you think over what you learned and prepare you for the next week. Theoretical information will be available from book ["Intro to Autonomous Mobile Robots"](#). You can also ask from forum.

You can install [ROBOTC 30 day trial](#) to your computer with ["Virtual World"](#) which enables you to test your function in your computer. You have to open from "Robot" menu "Global variables".

## Description

This homework gives you further experience in calculating robot's position in x,y plot and  $\alpha$  and  $\Delta x$  and  $\Delta y$ .

## Deliverable

Your target is to make your positioning program complete in the following way:

It is possible to write global variables:

```
float x_dest=25;//Destination on x axis in cm
```

```
float y_dest=25;//Destination on y axis in cm
```

Then you compile the program and download it to robot. Robot is facing x-axis and has to drive to destination controlled by speed  $v$  and rotational speed "omega". You may not tell the robot to drive by motor degrees!

Robot is MINDSTORMS base robot, distance between center point of wheels is 11.5 cm. Diameter of wheels is 5.44 cm.

# Week 10 - Line following using PID

## Practical assignment

### The Aim

Introduction to practical uses of [PID](#) algorithm

### You will learn to

1. Make a robot follow a line smoothly

### Tools needed

1. ROBOTC 3.x or newer
2. USB cable
3. LEGO robot

### Work description

1. Open line following sample program included in RobotC installation. File->Open sample. Try to make it work. You can see that by implementation it is a simple on/off controller that turns in one direction when the light sensor value is above certain limit and the opposite direction in the other case. Take this as a basis for your code.
2. Add P (progressive) control parameter to your code. Find good-enough value for its multiplier. You can find details from included PDF file.
- 3.
4. By now your robot should be able to follow a line a bit better than the initial code, but tends to "overshoot" curves.



Add D (differential) parameter to stabilize movement and try to make it work better than before.

- 5.
6. (Advanced) Add I (integral) component and see if it improves anything
- 7.
8. (Advanced) Calculate [PID](#) parameters mathematically (Ziegler-Nichols method)

## Hints

- You can see current light sensor values from robot directly (navigate around in its menus to find it)
- 
- By default your code works as fast as it can, normally there is no need to [run PID](#) control that fast. Add a small delay to your control cycle to improve that
- 
- It would be nice if you could change [PID](#) parameter values dynamically on a running program using the buttons on your robot. You can use `nNxtButtonPressed` variable

## Homework Assignment

### Help

You can install [ROBOTC 30 day trial](#) to your computer with "[Virtual World](#)" which enables you to test your code in your computer

## Description

In this homework you will complete the line following program you started with in a practice session.

## Deliverable

Make your robot follow a line using at least P and D [PID](#) parameters, you can find more details from the practice session guide. Test your algorithm using Virtual Worlds simulator from RobotC (Sensing->Robo Slalom II course).

# Week 11 - NAO - visual programming

## Getting started

Nao documentation:

<http://www.aldebaran-robotics.com/documentation/index.html>

## Download software

<https://dl.dropboxusercontent.com/u/76269978/NAO-software-bundle.zip>

Current version: 1.14.2

- Choregraphe suite
- Webots for NAO
- Python-2.7 SDK

<http://www.python.org/download/>

- Choose Python 2.7, 32 bit version (not Python 3, not 64 bit)

## NAOqi

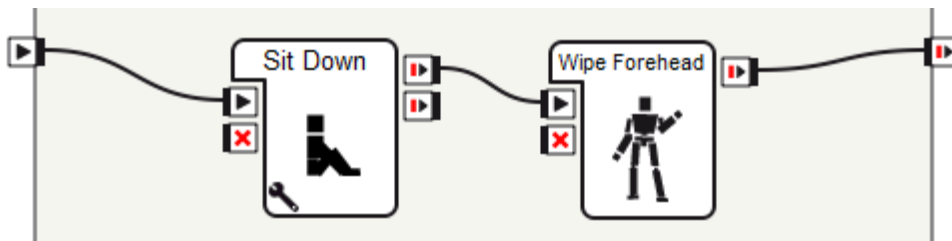
NAOqi is a framework that makes it possible to use functionalities on NAO through a client-server interface. Framework on a real robot will be loaded automatically during start-up and is accessible through a web interface [http://ROBOT\\_IP:9559](http://ROBOT_IP:9559). This framework can also be started on a local machine in order to simulate real NAO. Programs that want to communicate with NAO must create a connection with this framework.


## Visual programming: Choregraphe

<http://www.aldebaran-robotics.com/documentation/software/choregraphe/index.html>

A simple visual programming tool. By default creates a simple (no physics, robot is „floating“ in the air) simulation of Nao called local NAOqi.

Create a program by dragging blocks to canvas and connecting them:



Application starts by clicking: 

This behaviour is simulated locally in Choregraphe. To simulate gravity and environment an external simulator must be used. A real or simulated Nao can be connected by clicking Connection -> Connect to. All simulators **run** on a local machine with IP 127.0.0.1 (loopback interface) and have port number 9559 by default for communication. If one is detected by Choregraphe it should appear in a list displayed, if not IP and port must be entered manually. When communicating with real robot IP is the IP of the robot and port fixed to 9559.

## Simulation: Webots

[http://www.aldebaran-robotics.com/documentation/software/webots/webots\\_index.html](http://www.aldebaran-robotics.com/documentation/software/webots/webots_index.html)

Webots is a versatile robotics simulator that has a special version for NAO.

To start the simulation click [Run](#). Now you can try to connect from Choregraphe and [run](#) your behaviour. By default the simulator creates virtual naoqi that can be accessed using IP 127.0.0.1 (localhost) and port 9559.

If NAO fails to walk properly during the simulation (stops mid-step, stumbles) try a lower simulation timestep (from left menu WorldInfo -> basicTimeStep). You can also disable automatic stopping when the foot contact is lost by using the following Python commands:

```
motion = ALProxy("ALMotion", "127.0.0.1", 9559)
motion.setMotionConfig([[ "ENABLE_FOOT_CONTACT_PROTECTION",
False]])
```

### Python programming

This requires Python and NAOqi Python SDK to be installed.

<http://www.aldebaran-robotics.com/documentation/dev/python/index.html> - examples are provided throughout Nao documentation, you can also find them where you have installed naoqi-sdk (doc\examples\python).

If you know the IP and port of a running simulation or real robot you can send commands to it using Python scripts. Programming blocks in Choregraphe call python scripts, you can access them by double clicking on a block if the block contains no further sub-blocks inside it.

Create a file with .py suffix like my\_program.py. You can edit Python file by right clicking on it and choosing „Edit with [IDLE](#)“. You can execute your Python script by pressing F5 inside [IDLE](#).

A simple Python program for moving Nao's head would be:

```
from naoqi import ALProxy
motion = ALProxy("ALMotion", "127.0.0.1", 9559)
motion.setStiffnesses("Body", 1.0)
motion.angleInterpolation("HeadYaw", 1.5, 2.0, True)
motion.angleInterpolation("HeadYaw", 0.0, 2.0, True)
motion.setStiffnesses("Body", 0.0)
```

Line by line in words:

1. Import required libraries. Naoqi is library added by Python Naoqi SDK
2. Create proxy for communicating with modules on Nao. Proxy is an object that delegates communication. We have created proxy to ALMotion module that controls Nao joints and movements. 127.0.0.1 is an IP and 9559 port to connect to, in this case this would be a running simulator
3. We tell Nao to **turn** power on for all joints (1.0 maximum, 0.0 off). For values lower than 1.0 Nao may not be able to complete given tasks and may fall when trying to stand or walk. **IMPORTANT:** when writing a script always set joint stiffness to 1.0 before attempting any movement with Nao.
4. Set Head Yaw joint (head left-right movement) to 1.5 radians from the centre, this command must be executed in 2.0 seconds. True means that movement is absolute (0.0 radians is always fixed centre).
5. Move head back
6. Power joints off to save power (for real robot). **IMPORTANT:** make sure Nao is in a safe position before calling that

## Connecting to a real robot

Nao can be started by pressing and holding a button on its chest (about 4 seconds). Start-up might take over a minute, when it has finished Nao will give a verbal notification and LEDs on the feet should light up.

First connection must be created using a cable. Nao and controlling computer must be connected to the same network with IP addresses in the same subnet. Wired connection must have DHCP enabled (IP must be assigned automatically). If Nao is connected with a cable (port is in the back of robots head) it should obtain an IP address. Nao announces this address if the chest button is pressed.

Using a web browser navigate to `http://ROBOT_IP` to access visual interface. If there are any stronger wireless network nearby they should appear under „Networks“ . Wifi networks can be selected as 'default' so that Nao will try to connect to that network on next boot.

After the robot has an IP and computer is in the same subnet Choregraphe and Python programming tools can connect to a robot by using external IP instead of 127.0.0.1.

For other proxies and available methods see:

<http://www.aldebaran-robotics.com/documentation/naoqi/index.html>

## Practical assignment

### The Aim

The aim of this session is to get familiar with visual programming and simulation software used for Aldebaran Nao.

## You will learn to

1. Create simple behaviours in Choregraphe
2. Use simulation environment Webots
3. Upload and test your behaviour on a real robot
4. Create an animation with Choregraphe

## Tools needed

- Choregraphe software suite
- Webots for NAO simulator
- Nao robot

## Work description

### **1. Getting started with Choregraphe**

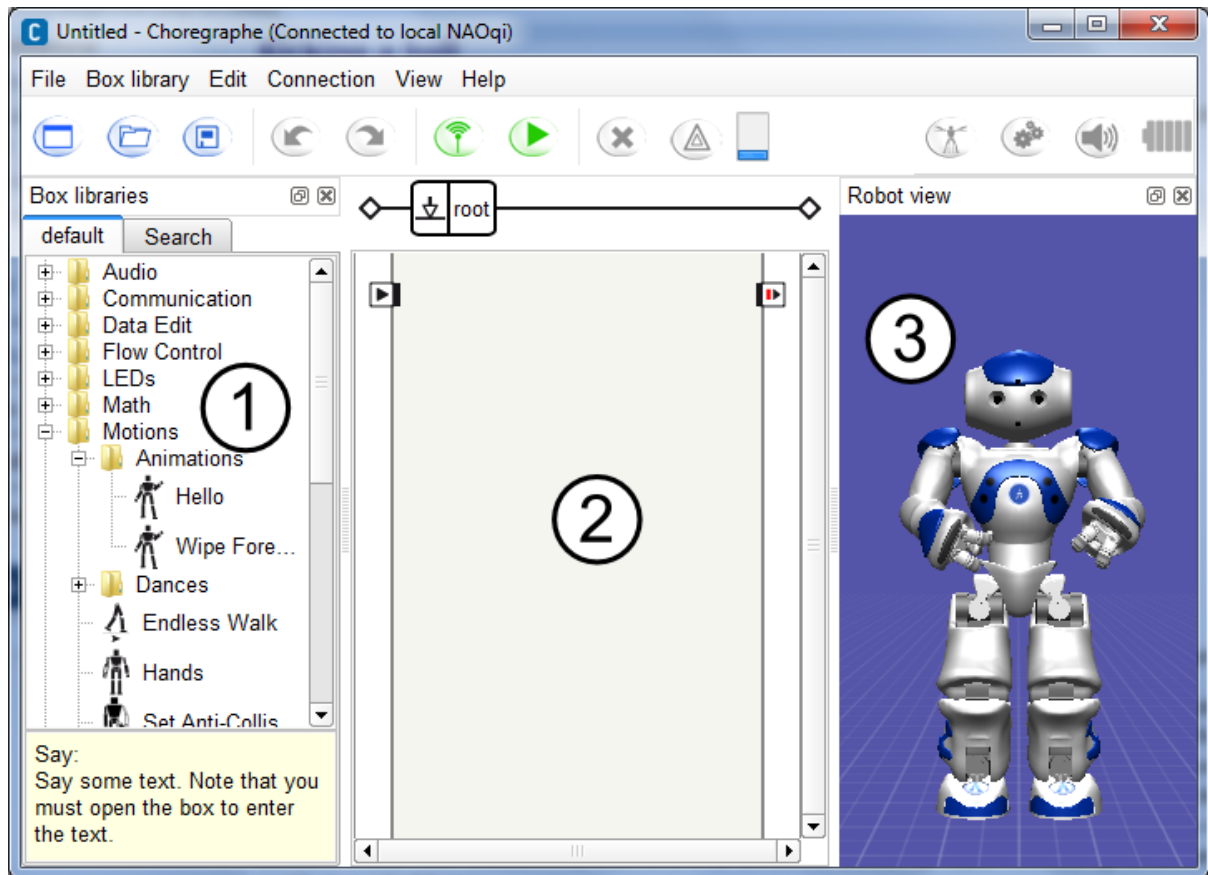
A simple visual programming tool. By default creates a simple (no physics, robot is „floating“ in the air) simulation of Nao called local NAOqi.

When you start the program you should see a 3D view of Nao ("Robot View"), if not make sure that View->Robot View is selected.

You can find general documentation and tutorials from:

<http://www.aldebaran-robotics.com/documentation/software/choregraphe>

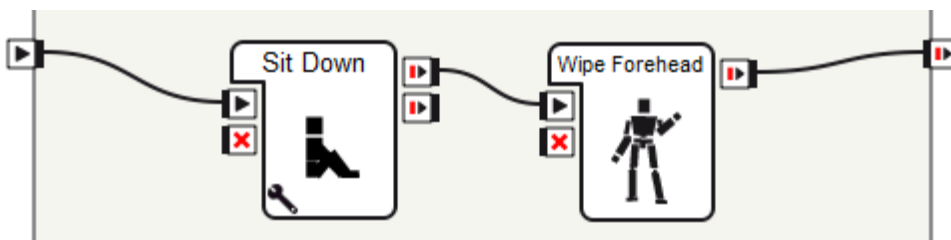





We will call the components in the following way:

1. Libraries
2. Canvas
3. 3D View

Create a program by dragging blocks to canvas and connecting them:



Application starts by clicking: 

Explore different functionalities the libraries offer. Try some of them out. 3D view of Nao can do some of these things, but not all. What are the limitations?

## **2. Getting started with simulations**

[http://www.aldebaran-robotics.com/documentation/software/webots/webots\\_index.html](http://www.aldebaran-robotics.com/documentation/software/webots/webots_index.html)

After starting the simulation you can try to connect to it with Choregraphe and `run` your behaviour.

A real or simulated Nao can be connected by clicking Connection -> Connect to in Choregraphe. All simulators `run` on a local machine with IP 127.0.0.1 (loopback interface) and have port number 9559 by default for communication. If one is detected by Choregraphe it should appear in a list displayed, if not IP and port must be entered manually. When communicating with real robot IP is the IP of the robot and port fixed to 9559.

Create a Choregraphe program that:

1. Makes Nao stand up
2. Walk 2m in forward direction
3. Wipe his forehead
4. And sit down again

All of these actions can be performed by adding blocks to the canvas and connecting them.

## **3. Running a behaviour on real Nao**

[http://www.aldebaran-robotics.com/documentation/getting\\_started/index.html](http://www.aldebaran-robotics.com/documentation/getting_started/index.html)

Nao can be started by pressing and holding a button on its chest (about 4 seconds). Start-up might take over a minute, when it has finished Nao will give a verbal notification and LEDs on the feet should light up.

In your practice sessions all Nao robot connect automatically to the Wifi network you can get their IP if you push the chest button for a few seconds. In order to communicate with Nao you must connect to the same network, ask your supervisor for further details when and where to connect.

Once you are connected you should be able to connect to a real robot using its IP like you did with the simulator.

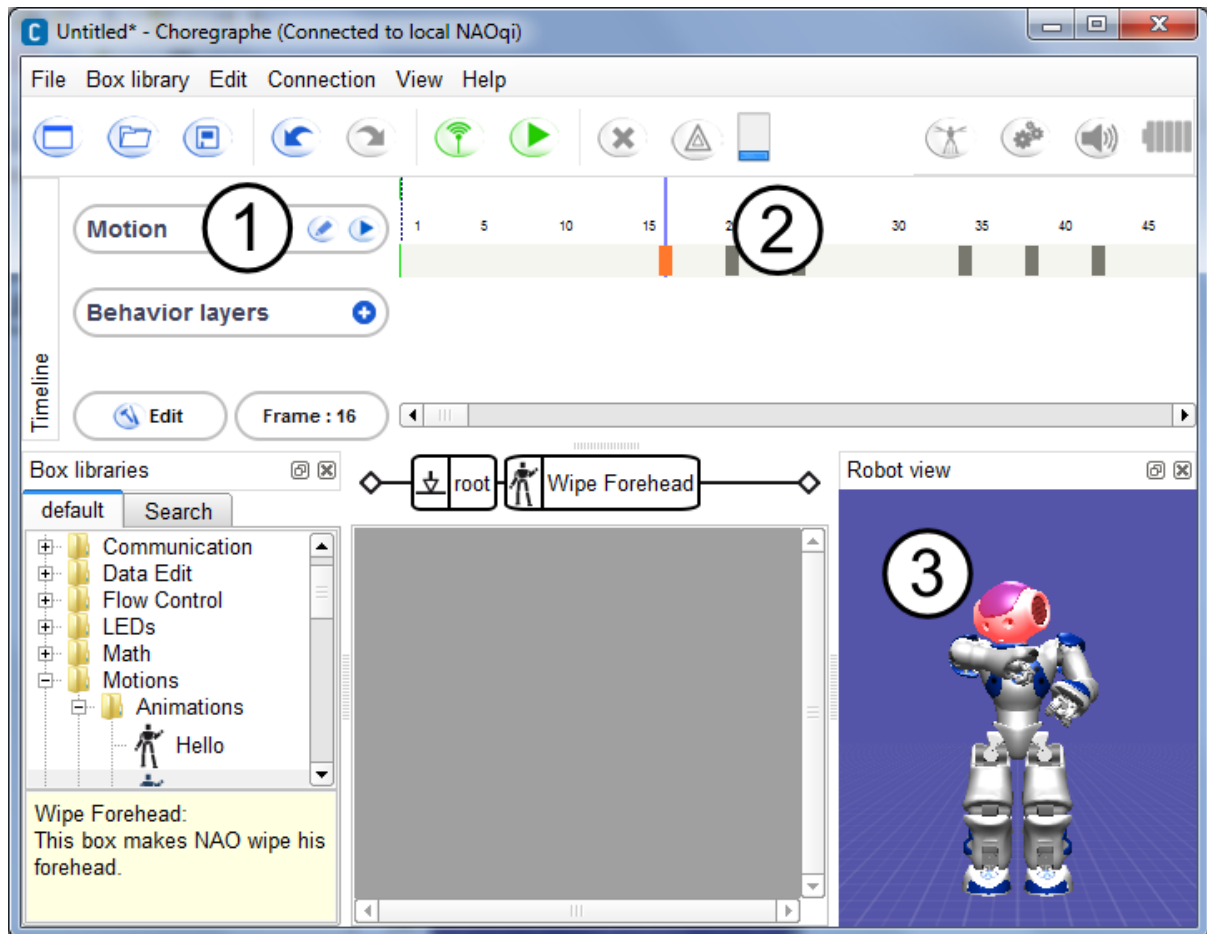
Click 'Play' to automatically upload your current program to Nao and start it. Once you are done disconnect.

#### **4. Creating animations**

We will take a look into the most useful feature in Choregraphe - animating.

Please make sure you are not connected to a real robot anymore as the following will make it fall. Click on Nao joints on 3D view, a small window will popup that enables you to change angles for the joints. Move some them. Now connect to a simulator, make Nao stand up and try out what it takes to make the simulated robot fall by moving its legs.

Now lets take a look how we can make an animation using this tool. Drag "Wipe forehead" block on your canvas. Double-click it to see what is inside.



1. This represents the motion layer. You can click on the pen icon to see it in detail (for each joint)
2. Keyframes this is more compact view from the timeline that opened from the pen icon. Each marker means that at this point in time we have defined some specific target angle for the joints Nao has to move. By clicking on these you should see Nao moving in the 3D view (if not connect to localNaoqi). Horizontal numbers you can see are not seconds, but frames. By default 25 frames equal one second. So the closer keyframes are the faster is the transition between them (be careful with that).
3. 3D view reacts to the changing the keyframe. You can modify keframe by clicking on joint and setting a new value

and then right-clicking on a keyframe and choosing "Store joints in keyframe".

Create an animation of your own (be creative, but remember that to try it on real robot it also has to be safe). Start up your simulator again and test it out. Does it make Nao fall or behave unexpected?

To get your animations working you need to add "Motor on/off" blocks before and after it (stand up and sit down blocks do this for you).

We need to tell Nao to **turn** power on for all joints.

**IMPORTANT:** when creating your own animations always **turn** motors on before attempting any movement with Nao.

After the animation power joints off to save power (for real robot).

**IMPORTANT:** make sure Nao is in a safe position before calling that

If there is time you can try your animation on real robot.

## Homework Assignment

### Description

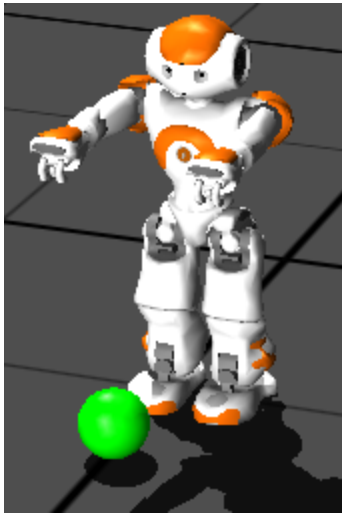
This homework gives you further experience in creating animation in Choregraphe and also maintaining Nao balance when doing them.

### Deliverable

1. Install software and setup project

Download and install Webots for NAO and Choregraphe (you can get access to this software through your supervisor).

Download "Homework Webots project" from the course materials and open in with Webots (File->Open World). You will then see the following scene:



## 2. Make Nao kick the ball

Create an animation that makes Nao kick the ball while standing (do not just walk into the ball). This requires you to balance the robot on one foot while moving the other. After the kick Nao should take back its initial position. The robot must not fall while doing this.

Upload your Choregraphe project file to the repository. The style, efficiency and safety will be evaluated.

# Week 12 - NAO - Python scripts

## Practical assignment

### The Aim

Getting familiar with controlling Nao using Python scripts. Building a basis for the next homework.

### You will learn to

1. Send simple commands to Nao using Python
2. Use Nao camera and process its output

### Tools needed

1. Python 2.7
2. NaoQi library for Python
3. Webots simulator
4. Nao robot

### Work description

#### **1. Getting started with Python**

If you know the IP and port of a running simulation or real robot you can send commands to it using Python scripts. Programming blocks in Choregraphe call python scripts, you can access them by double clicking on a block if the block contains no further sub-blocks inside it.

Create a file with .py suffix like my\_program.py. You can edit Python file by right clicking on it and choosing „Edit with **IDLE**“. You can execute your Python script by pressing F5 inside **IDLE**.

A simple Python program for moving Nao's head would be:

```
from naoqi import ALProxy
motion = ALProxy("ALMotion", "127.0.0.1", 9559)
motion.setStiffnesses("Body", 1.0)
motion.angleInterpolation("HeadYaw", 1.5, 2.0, True)
motion.angleInterpolation("HeadYaw", 0.0, 2.0, True)
motion.setStiffnesses("Body", 0.0)
```

Line by line explanation in words:

1. Import required libraries. Naoqi is library added by Python Naoqi SDK
2. Create proxy for communicating with modules on Nao. Proxy is an object that delegates communication. We have created proxy to ALMotion module that controls Nao joints and movements. 127.0.0.1 is an IP and 9559 port to connect to, in this case this would be a running simulator
3. We tell Nao to **turn** power on for all joints (1.0 maximum, 0.0 off). For values lower than 1.0 Nao may not be able to complete given tasks and may fall when trying to stand or walk. **IMPORTANT**: when writing a script always set joint stiffness to 1.0 before attempting any movement with Nao.
4. Set HeadYaw joint (head left-right movement) to 1.5 radians from the centre, this command must be executed in 2.0 seconds. True means that movement is absolute (0.0 radians is always fixed centre).
5. Move head back
6. Power joints off to save power (for real robot). **IMPORTANT**: make sure Nao is in a safe position before calling that



Get this program working on your simulator. What should we change to make it work on real Nao?

You can read more about different proxies and functions you can call from:

<http://www.aldebaran-robotics.com/documentation/naoqi/>

## 2. Getting images from Nao camera

Find a proxy for video from Nao documentation and learn how to use it.

A major problem that you are about to discover is that image data from Nao camera is a string instead of a convenient matrix used in NXT exercises. Or to be more precise it is a raw image buffer data data that contains unsigned 8 bit integers that are interpreted as a string by Python. Fortunately there is a way to use this data to create an image that was used in old OpenCv Python bindings and then convert it to new familiar format that enables to reuse colour detection from NXT code.

The following code converts Nao camera image to OpenCV type image we used before:

```
import cv2
import numpy

naoimg = ...

# RGB image container. 320x240. 8 bits each value (unsigned), 3
channels

cv_img = cv2.cv.CreateImageHeader((320,240), cv2.cv.IPL_DEPTH_8U,
3)

# naoimg is an array. Image data is result[6] as string.
# Last parameter needs to be width*channels
cv2.cv.SetData(cv_img, naoimg[6], 320*3)
```

```
# Convert to OpenCv2 compatible format
img = numpy.asarray(cv2.cv.GetMat(cv_img))
```

Your task is to figure out how to get variable 'naoimg' which should be camera frame from Nao. You can test your code with the simulator. Try to display result on screen like with colour detection exercises.

### **3. Detect a colour**

Try to get your colour detection algorithm working with Nao camera.

## **Homework Assignment**

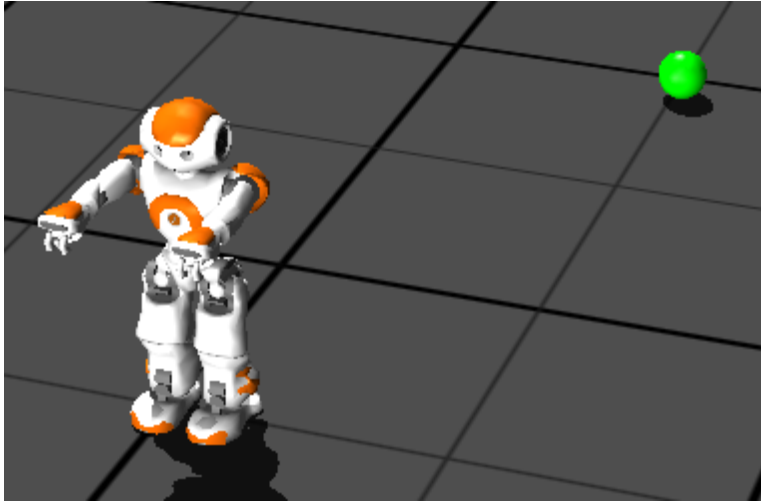
### **Description**

In this homework you will learn how to navigate with Nao to a destination using camera as feedback.

### **Deliverable**

#### **1. Setting up the environment**

Download Webots project II from the course materials. You will see the following scene:



In this setup Nao will be standing on a floor with his back turned to a ball, the ball is within a radius of  $\sim 2\text{m}$ .

2. Stand up (needed only for real robot, you can make NAO fall to simulate this in simulator)

First, make Nao stand up (find and read about ALRobotPosture).

3. Make Nao walk to the ball

This task involves processing the camera input to find the ball (you can choose from two cameras Nao has) and sending commands to ALMotion proxy. Figure out how to use this function to make Nao **turn** and walk forward.

#### Hint

You will likely need to know the FOV, height and orientation of Nao cameras

[http://www.aldebaran-robotics.com/documentation/family/nao\\_h21/index\\_h21.html](http://www.aldebaran-robotics.com/documentation/family/nao_h21/index_h21.html)

Nao should be able to stop within the theoretical kicking radius  $< 30\text{cm}$  (this exercise does not require you to kick the ball, but be ready for that)

in the future). Right now the important part is to get the robot near the ball.

## Evaluation

The following factors are considered while grading your work:

- precision of reaching the goal (how close you get, do not need to directly face the ball)
- repeatability (how error prone the algorithm is). Remember that in the real world Nao may not be able to always walk a straight line, find ways to compensate this.
- code quality. A minor factor, but it has to be readable and reasonably commented
- Since walking algorithm is not your code do not worry if simulated Nao falls sometimes

# Week 13 - NAO - football exercise final

## Practical assignment

### The Aim

Is to get Nao robot walk to a ball and kick it in simulated and real environments.

### You will learn to

Recognise the differences between simulation and real environments

### Tools needed

1. Python 2.7
2. Naoqi Python libraries
3. Webots for NAO simulator
4. Nao robot

### Work description

1. Test your homework on real Nao

While testing on real Nao please make sure that:

1. You are ready to catch it if robot should lose its balance or power
2. You make Nao sit down and **turn** off the joint stiffness after your program completes

2. Combine walking to a ball and kicking it

If your "finding a ball" algorithm is working well enough try make Nao kick the ball after walking to it.

# Homework Assignment

## Description

In this homework you will learn to combine walking to a ball and kicking it.

## Deliverable

Use Choregraphe to upload your kicking behaviour to robot/simulator (View->Behavior Manager), then you can call in Python using ALBehaviorManager proxy. Improve your kicking and walking to work together.

## Evaluation

Will be graded during the next practice class. It is required to show the final solution on a real robot (make sure you can setup your color detection parameters fast and easy). Students who can not partake in the practice session can submit a video of their solution.

## Week 14 - NAO - final week

We have come to an end of the course at this point. We can use the final week for polishing and evaluating our solutions and thinking back at the course.

We would hereby like you to fill in the course feedback form to help us build a better course for next year.

## Appendix I - Top tips for solving issues

In this section we will give solutions to two issues that you are rather likely to run into while completing the course.

### Camera synchronization

An access to cap object (through grab, retrieve) is not itself synchronized in this example. If necessary (images appear corrupted) a lock (threading.lock) should be added around these methods. Locking is not used here as it is slow and having none has not caused problems during testing.

```
'''A simple example of taking individual frames from camera. Since images taken after longer period of time are out of sync (several seconds behind since buffer is not cleared very often) a special thread has been added to keep the buffer state fresh while main thread queries an image'''
```

```
import cv2
```

```
import thread
```

```
import sys
```

```
import time
```

```
import datetime
```

```
...
```

```
This works simultaneously with main thread and cleans camera frame buffer so a call to retrieve()
```

```
could get the latest image
```

```
...
```

```
def videoThread(cap):
```

```
    while(True):
```



```
    # Captures frame from camera, does not process it (also  
cleans buffer)
```

```
    cap.read()
```

```
    time.sleep(0.1)
```

```
# Create connection with attached camera
```

```
cap = cv2.VideoCapture(1)
```

```
thread.start_new_thread(videoThread, (cap,))
```

```
while(True):
```

```
    date = datetime.datetime.now()
```

```
    # Decodes and retrieves the frame captured by grab()
```

```
    flag, img = cap.retrieve()
```

```
    # Save image on disk with datetime as title
```

```
    cv2.imwrite((str(date)).replace(":", "_") + ".jpg",img)
```

```
    time.sleep(5)
```

```
    print str(date)
```

## Bluetooth connection issues

When using `nxt-python` library and connecting with bluetooth the created connection may not terminate and therefore make it impossible to connect to a robot for a period of time. This is due to a fact that Bluetooth communication port is not closed, unfortunately it is not possible to make the library reuse already created connection (on Windows). Port will only be available again after a timeout occurs and it is released by the operating system.

One way to prevent this from happening is to close Bluetooth connection in your code:

```
import nxt
b = nxt.locator.find_one_brick()
try:
    # Write your code here
finally:
    # This part always executes
    b.sock.close()
```

If anything happens inside try block (exception, keyboard interrupt etc) or if the code just finishes its execution `b.sock.close()` is called that terminates Bluetooth connection that was created by the `find_one_brick()` function.