# Assignment 1: Control and obstacle avoidance

ETH Zurich

Individual Demonstration: Thursday, 09.10.2013 at 15:15
Individual Software Due: Thursday, 09.10.2013 at 23:00
Group Work Demonstration: Thursday, 16.10.2013 at 15:15
Group Work Software Due: Thursday, 16:10.2013 at 23:00

It's time to put your new robot into use. You would like your robot to go to places as you command, such as fetching beer from the refrigerator. Unfortunately, your apartment is not empty. You don't want to have to tell your robot how to avoid every wall and furniture that may lay in the robot's way; instead, you would like your robot to use its sensor to avoid them as it sees necessary.

## 1 Hardware

### 1.1 Thymio-II

Thymio-II [4] is a small differential-drive robot with a large number of sensors and actuators, as shown in Figure 1. The robot is programmable using Aseba - an event-based architecture. The robot comes with pre-programmed behaviours, namely:

- Friendly (green) - Thymio-II follows the object in front of it.
- Explorer (yellow) - Thymio-II explores the environment while avoiding obstacles.
- Fearful (red) - Thymio-II detects shocks and free falls, and shows the direction of gravity.
- Investigator (cyan) - Thymio-II follows a trail. The trail must be at least 4 cm wide with high contrast, the best is black on white.
- Obedient (purple) - Thymio-II follows the commands from the buttons or from a remote control.
- Attentive (blue) - Thymio-II responds to sound. We can control the robot by clapping. 1 clap = go straight ahead / turn. 2 claps = run / stop. 3 claps = turns in a circle.

Play with these pre-programmed behaviours to better understand how different sensors and actuators of Thymio-II work.

### 1.2 Carmine 1.09 Sensor

Carmine 1.09 is an RGB+D camera that gives both color and depth information of an environment. It projects infrared structured light into the environment and calculates the depth by analyzing the distortion pattern of the structured light. In addition, it has a standard image camera to capture color information of the scene. The sensor provide an RGB+D image by registering the depth and color images.

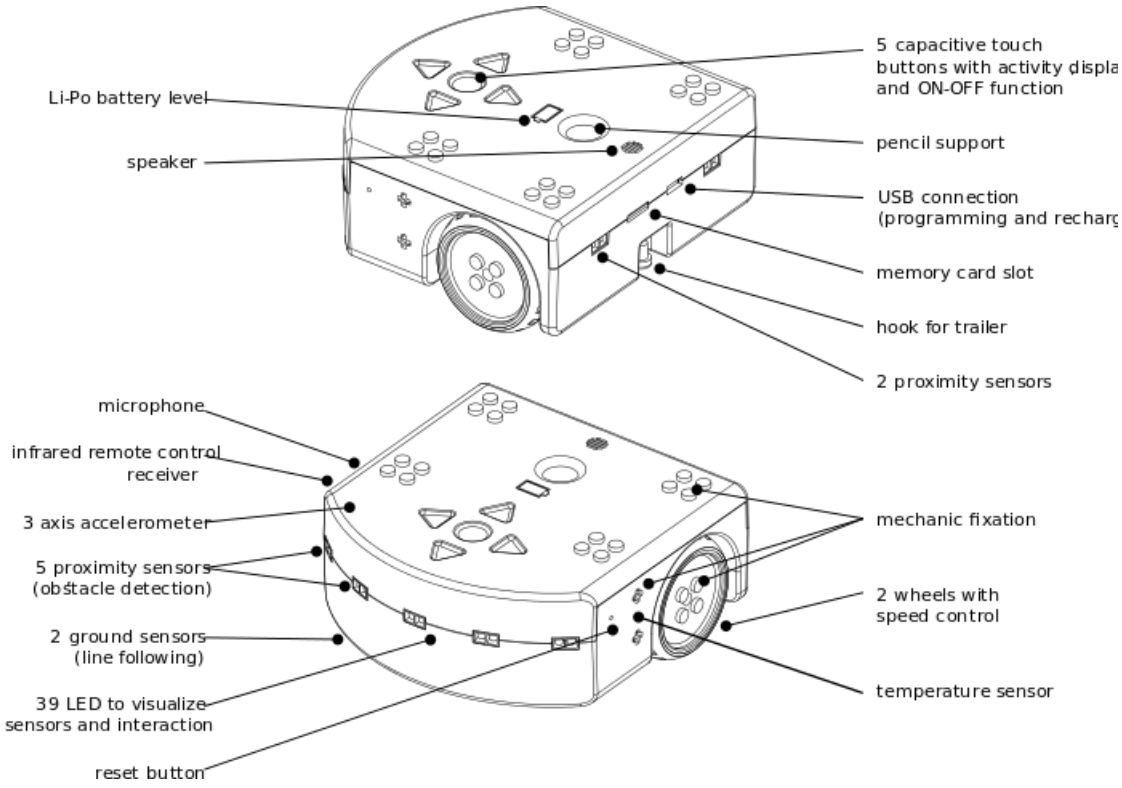


Figure 1: Sensors and actuators of Thymio II

# 2 Software

Software installation guide is available here: http://www.roboscoop.org/docs. Follow the installation guide and install all the necessary software.

## 2.1 Ubuntu

Ubuntu is a Debian-based Linux operating system and is the officially supported operating system of ROS. For this course we recommend you to use Ubuntu 14.04 LTS (Trusty). Free distribution of Ubuntu 14.04 can be found here: http://releases.ubuntu.com/14.04/.

## 2.2 ROS

ROS (Robot Operating System) [5] provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more.

## 2.3 OpenNI SDK

OpenNI is an open-source framework for "natural interaction". It provides the driver for Carmine 1.09 sensor and additional libraries.

## 2.4 EiffelStudio

EiffelStudio [6] is an interactive development environment, specifically crafted for the Eiffel method and language. Download a free version of **EiffelStudio 14.05** for Linux from SourceForge: http://sourceforge.net/projects/eiffelstudio/files.

## 2.5 Roboscoop

Roboscoop is a robotics framework built on top of Simple Concurrent Object Oriented Programming (SCOOP). The source code of Roboscoop is split into three parts: ROS package `roboscoop_ros`, which is responsible for external communication and external execution (robot's software, reused ROS functionality), Eiffel project of Roboscoop library (`roboscoop_lib` - top level of Roboscoop) and Eiffel project `roboscoop_app` where you can create your applications.

## 2.6 SVN

Subversion [11] is an open-source version control system, which allows you to keep old versions of files and directories (usually source code), keep a log of who, when, and why changes occurred, etc. You will submit your work using an SVN repository. The link will be sent to you via email in the second week of the semester.

# 3 Tutorials

At the end of these tutorials, you should be able to communicate with Thymio II using Roboscoop. In particular, you should be able to write code to activate and receive information from all sensors and actuators of Thymio II.

### 3.1 Ubuntu

If you have never used a Linux operating system, get yourself familiar with the basics of Linux. Internet has numerous sites and videos that explain Linux operating system in general and Ubuntu in particular. Some recommended introductions include the official Ubuntu documentation and "Introduction to Linux" by Garrels [8].

### 3.2 ROS

Read the ROS tutorials [9]. Follow the beginner level tutorials and perform the suggested exercises. By the end of the tutorials, you should understand the difference between topics and services and be able to write a publisher and subscriber node and a service and client node.

TF and RViz packages are useful packages for this class. Follow the tutorials for these packages and learn how they work.

### 3.3 Eiffel and SCOOP

Eiffel is an object-oriented language with design by contract and is the base language for SCOOP (Simple Concurrent Object-Oriented Programming), on which Roboscoop builds. There is a list of resources for learning Eiffel. In particular, an online tutorial of Eiffel is available here.

The SCOOP concurrency model seeks to remove the gap between the object-oriented concepts and the techniques used for handling the multithreaded or concurrent parts. It brings to the world of concurrency the same systematic O-O development techniques that have made their mark in the sequential world. Concurrent Eiffel with SCOOP contains a short description of SCOOP, and "Practical framework for contract-based concurrent object-oriented programming" by Nienaltowski [10] contains a more thorough description.

### 3.4 Aseba and asebaros

Aseba [7] is a set of tools used to program Thymio II. Aseba is an event-based architecture for real-time distributed control of mobile robots and targets integrated multi-processor robots or groups of single-processor units. The Aseba language is described in `https://aseba.wikidot.com/en:asebalanguage`, and the programming interface is found in `https://aseba.wikidot.com/en:thymioapi`.

Write a program in Roboscoop that can receive information from various sensors and activate various actuators of Thymio II and ensure that every sensor and actuator is working.

### 3.5 SVN

Subversion keeps a single copy of the master sources, called the source "repository". The source repository contains all the information to permit extraction of previous versions of its files at any time.

Some basic commands of SVN are as follows:

- Update: "Update" brings changes from the repository into the working copy.

```
svn update
```

- Commit: "Commit" sends changes from your working copy to the repository.

```
svn commit -m "... commit message ..."
```

- Add: "Add" puts files and directories under version control, scheduling them for addition to the repository. These files and directories will be added in next commit.

```
svn add PATH...
```

- Delete: "Delete" removes files and directories from version control.

```
svn delete PATH...
```

- Help: "Help" brings up a list of commands.

```
svn help
```

In general, you will `svn update` to get the latest version controlled files and `svn commit` to commit your latest changes. You *must commit* to send the local changes to the repository.

When you create a new file, you must `svn add` the file for the file to be under version control. Any file that is in your local svn directory but is not version controlled cannot be viewed by other users of the repository. Likewise, any file that is locally deleted but not `svn delete`'d is still viewable by other users. Do not forget to svn add files that you want us to see and delete files that you do not want to be part of your assignment submission.

# 4 Robot control

## 4.1 Background

The main objective of robot control is to calculate solutions for the proper corrective action from the controller that result in system stability, that is, once the system reaches the target point, the system will stay within a certain distance from the target point without oscillating around it. Control algorithms can be open loop or closed loop (feedback). In open loop control, you control the robot without any feedback from the internal or external sensors. Feedback control, on the other hand, uses internal or external sensing to determine the current error between the actual and desired state of the robot.

Proportional-Integral-Derivative (PID) control [1] is a popular feedback control design. The proportional term is proportional to the error between the desired and actual system outputs and controls how quickly the robot reacts to the error. The integral term is proportional to both the magnitude of the error and the duration of the error and accelerates the movement of the process towards the desired output and eliminates the residual steady-state error. The derivative term is proportional to the slope of the error over time and improves settling time and stability of the robot.

Mathematically, the PID controller is

$$u(t) = K_p\, e(t) + K_i \int_0^t e(\tau)\, \mathrm{d}\tau + K_d \frac{\mathrm{d}}{\mathrm{d}t} e(t) \tag{1}$$

where $u(t)$ is the control output; $K_p$, $K_i$, and $K_d$ are control gains for the proportional, integral, and derivative terms; $e$ is the error between the desired value and measured value; $t$ is the current time; and, $\tau$ is the total time from time 0 to the current time $t$.

We can make a robot go to a desired position(goal) by controlling its heading $\theta_{robot}$ towards the goal, as shown in Figure 2. Given the robot's position $(x_r, y_r)$ at time $t$, the heading error $\theta_{error}$ between the robot and the goal position $(x_g, y_g)$ is

$$\theta_{error} = \theta_{goal} - \theta_{robot} = \arctan(\frac{y_g - y_r}{x_g - x_r}) - \theta_{robot}. \tag{2}$$

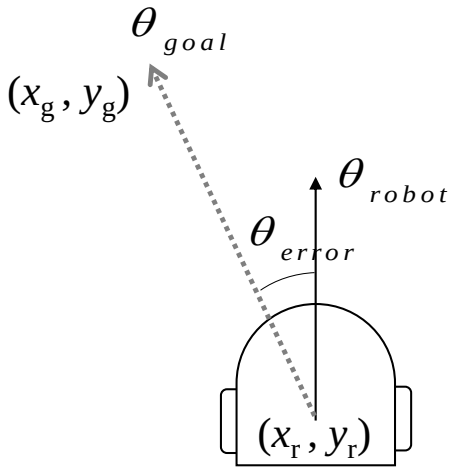


Figure 2: Control output $\theta$

### 4.2 Task

Write a PID controller for Thymio II in Roboscoop to control its heading. Your code should take desired goal position as input and control the heading towards the goal using the PID controller.

#### 4.2.1 Hints

- Robot odometry: `roboscoop_thymio_navigation_driver.py` publishes robot's odometry at time $t$ and sends velocity command $(v_x, v_z)$ to the robot. Your job is to send appropriate velocity commands based on the robot's odometry.
- Goal tolerance: Consider including a threshold such that the robot stops moving when it is within a threshold of the goal.
- Linear velocity: Consider making the linear velocity depend on the angular velocity. This will make the robot slow down when it needs to make a big turn and have a smaller turning radius.

## 5 Obstacle avoidance

### 5.1 Background

Obstacle avoidance is the process of satisfying a control objective without colliding into obstacles. Obstacle avoidance should be written such that in the case of obstacles, the robot makes appropriate motions around the obstacles while trying to achieve the goal following the shortest path. If there are no obstacles, the robot should move directly towards the goal location.

TangentBug [2] is an obstacle avoidance algorithm in the Bug [3] algorithm family. The Bug algorithms combine local planning algorithms with global planning algorithm. With a minimal introduction of a global model, the Bug algorithms ensure that the purely reactive local planning algorithms can converge to the desired goal globally. TangentBug is a Bug algorithm, specifically-designed for range data. The basic idea behind the algorithm is as follows:

1. Move toward the goal $T$ until:
   - If the goal $T$ is reached, stop.

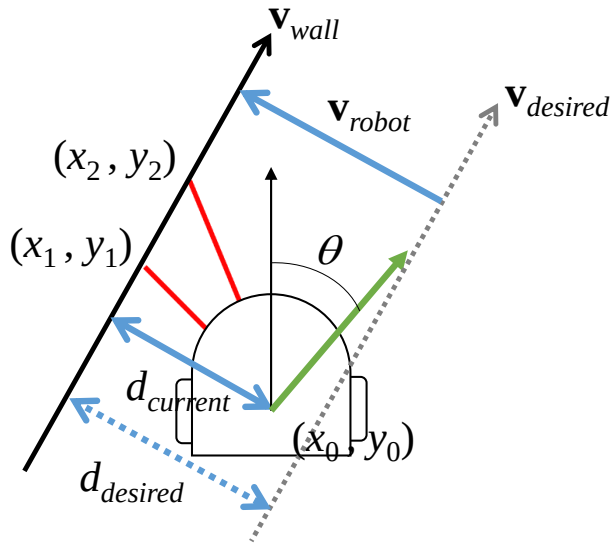


Figure 3: Control output $\theta$ for boundary following

- If an obstacle in the direction towards to goal is detected, go to step 2.

2. Choose a boundary-following direction and move along the obstacle while recording the minimum distance $d_{min}(T)$ to the goal $T$ until:
   - The goal is reached. Stop.
   - If the leave condition holds, i.e., the robot see no obstacle at $V_{leave}$ such that $d(V_{leave}, T) < d_{min}(T)$, go to step 3.
   - If the robot has completed a loop around the obstacle, stop and report that the target is unreachable.
3. Perform the transition phase. Move towards $V_{leave}$ until reaching a point $Z$ such that $d(Z, T) < d_{min}(T)$, then go to step 1.

In boundary-following, the goal is to keep robot's heading parallel to the wall while at the same time keeping the robot a constant distance away from the wall. Figure 3 shows the basic concept behind boundary following. Taking $(x_1, y_1)$ and $(x_2, y_2)$ as the two closest sensor values to the robot, we can estimate the closest wall $\mathbf{v}_{wall}$ as

$$\mathbf{v}_{wall} = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}. \tag{3}$$

The vector perpendicular to the wall is then

$$\mathbf{v}_{robot} = \begin{pmatrix} y_2 - y_1 \\ -(x_2 - x_1) \end{pmatrix}. \tag{4}$$

We can now calculate the distance $d_{current}$ from the robot to the wall $\mathbf{v}_{wall}$ as a projection of the vector $\mathbf{v}_{sensor} = \begin{pmatrix} x_1 - x_0 \\ y_1 - y_0 \end{pmatrix}$ from the robot origin $(x_0, y_0)$ to a point $(x_1, x_2)$ on the wall $\mathbf{v}_{wall}$ to a unit vector $\hat{\mathbf{v}}_{robot}$, i.e.,

$$d_{current} = |\hat{\mathbf{v}}_{robot} \cdot \mathbf{v}_{sensor}| = \frac{(y_2 - y_1)(x_1 - x_0) - (x_2 - x_1)(y_1 - y_0)}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}. \tag{5}$$

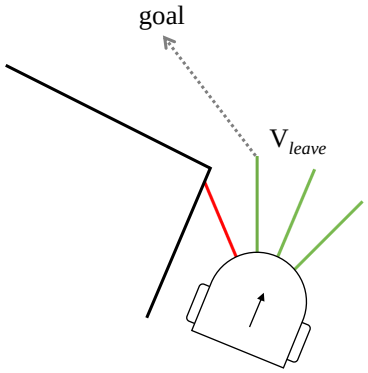


Figure 4: Transitioning to the $V_{leave}$

Given the desired distance $d_{desired}$ between the wall and the robot, we can now compute the heading $\theta$ for the robot from $\hat{\mathbf{v}}_{wall}$ and $\hat{\mathbf{v}}_{robot}$ as

$$\theta = \arctan(\frac{\mathbf{v}_{\theta,y}}{\mathbf{v}_{\theta,x}}) - \theta_{robot}, \tag{6}$$

where

$$\mathbf{v}_\theta = d_{desired}\hat{\mathbf{v}}_{wall} + (d_{current} - d_{desired})\hat{\mathbf{v}}_{robot} \tag{7}$$

Note that if the vectors are computed with respect to the robot's coordinate frame, then $\theta_{robot}$ is zero.

At some point, there is no more obstacle on the robot's way to the goal. The location in which the robot can go out of the obstacle avoidance is $V_{leave}$ as shown in Figure 4. This exit point $V_{leave}$ is the first free space that ensures that the robot is closer to the goal than it has been, i.e., $d(V_{leave}, T) < d_{min}(T)$. Once the robot moves to this exit position $V_{leave}$, it will be closer to the goal.

## 5.2 Task

Write TangentBug obstacle avoidance algorithm in Roboscoop. The TangentBug algorithm requires three distinctive behaviors: going to the goal, avoid obstacle (following a wall), and transitioning from obstacle avoidance to going to the goal. Optionally, change robot's LEDs color to indicate the current state/behavior.

## 5.3 Hints

Thymio's range sensors are limited in number and range. Consider the following modifications to the original TangentBug algorithm.

- **Sensor calibration**: Be sure to calibrate properly all the sensors before starting the controller implementation.
- **Obstacle detection**: The original algorithm is designed for a point robot, but Thymio has volume. A minimum of three sensors must be unblocked in the direction of travel for Thymio to travel safely.
- **Obstacle avoidance**: In the obstacle avoidance mode select different distance thresholds for each sensor (when detecting an obstacle) or a unique threshold (if it exists) that you are sure is inside the working range of all the sensors. This is because the detection range for one sensor can be much lower than the others. Therefore, you must always ensure

that all the thresholds you set/use for the control are inside the working range of all the sensors that you use to control the device. Otherwise, if the controller uses a threshold that is outside the range of one/more sensors, then the controller is going to behave in crazy ways and the controller would be unstable or not robust.

- **Unreachable goal**: Loop closure is a difficult problem and requires tracking of robot's trajectory and good sensory information. Consider a simple solution in which your algorithm detects when the robot comes near the starting point of obstacle avoidance after it has been significantly away from it.

# 6 Grading

## 6.1 In-class demonstration (20 points)

You will start at $(0, 0)$ and be given a goal point $(x, y)$ between $(-1m, -1m)$ and $(1m, 1m)$ to reach.

### 6.1.1 Individual demonstration (10 points): Thursday, 09.10.2013 at 15:15

In the individual demonstration, you will demonstrate how your accurately your robot reaches a goal.

- Accuracy (8 points)
  - Within $1cm$ of the most accurate robot: 8 points
  - Every $1cm$ thereafter: -0.5 point
- State indicator (2 points): Change the robot's color to indicate its state
  - Yellow for go to goal
  - Green for at goal
- You will get 2 tries. Every extra try will cost 1 point.

### 6.1.2 Group demonstration (10 points): Thursday, 16.10.2013 at 15:15

In the group demonstration, you will demonstrate as a group how your group's robot reaches a goal while avoiding obstacles.

- Accuracy (5 points)
  - Within $2cm$ of the most accurate robot: 5 points
  - Every $2cm$ thereafter: -0.5 point
- Wall following (2 points)
  - Wall following more than $1m$: 1 point
  - No bumping: 1 point
- Transition to goal (2 points)
  - Transition from follow wall: 1 point
  - Transition back to follow wall: 1 point
- State indicator (1 point): Change the robot's color to indicate its state

  - Yellow for go to goal
  - Red for boundary following
  - Blue for transition to goal
  - Green for at goal
  - Purple if goal is unreachable

- You will get 2 tries. Every extra try will cost 1 point.

### 6.2 Software quality (20 points)

On the due date at 23:00, we will collect your code through your SVN repository. Every file that should be considered for grading must be in the repository at that time. Note that EIFGENs folder in your project contains auxiliary files and binaries after compilation. Please, DO NOT include EIFGENs folder into your svn repository.

#### 6.2.1 Individual evaluation (10 points): Thursday, 09.10.2013 at 23:00

- Choice of abstraction and relations (3 points)
- Correctness of implementation (4 points)
- Extendibility and reusability (2 points)
- Comments and documentation, including "README" (1 points)

#### 6.2.2 Group evaluation (10 points): Thursday, 16:10.2013 at 23:00

- Choice of abstraction and relations (3 points)
- Correctness of implementation (4 points)
- Extendibility and reusability (2 points)
- Comments and documentation, including "README" (1 points)

## References

[1] Astrom, K., and Murray, R. 2008. Chapter 10: PID Control. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press.

[2] Kamon, I., Rimon, E., and Rivlin, E. 1998. TangentBug: A Range-Sensor-Based Navigation Algorithm. *The International Journal of Robotics Research*. 17(9):934-953.

[3] Lumelsky, V. J., and Stepanov, A. A. 1897. Path-planning strategies for a point mobile automaton moving amidst obstacles of arbitrary shape. *Algorithmica*. 2:403-430.

[4] http://www.thymio.org

[5] http://www.ros.org

[6] http://docs.eiffel.com

[7] https://aseba.wikidot.com

[8] Garrels, M. *Introduction to Linux*. Fultus Corporation. 2010.

[9] http://wiki.ros.org/ROS/Tutorials

[10] Morandi, Benjamin. *Prototyping a concurrency model*. PhD Dissertation, ETH Zurich, 2014.

[11] http://subversion.apache.org/