System for robotic e-learning

SyRoTek

Practical Guide to the SyRoTek System

ver. 1.0

CTU FEL, Intelligent and Mobile Robotics Group
Contents

1 Introduction ........................................ 7

2 Web interface ..................................... 8
   2.1 Registration .................................. 8
   2.2 Login .......................................... 9
   2.3 Change the password ......................... 10
   2.4 Password retrieval ......................... 10
   2.5 Course enrollment ......................... 11
   2.6 Task selection ............................... 11
   2.7 Reservation .................................. 12
   2.8 Active reservation ......................... 16
   2.9 Invite a user ................................ 16
   2.10 Arena view ................................... 17
   2.11 Arena Control Panel ....................... 17

3 Command Line Interface ......................... 22
   3.1 Connecting to the server ................... 22
   3.2 syr_control .................................. 22
   3.3 ROS Launch file ............................... 23
   3.4 Simple ROS program - Braitenberg vehicle 25
      3.4.1 Introduction ............................ 25
      3.4.2 Braitenberg Vehicle ..................... 26
      3.4.3 Algorithm ................................ 26
      3.4.4 Processing inputs and outputs .......... 26
      3.4.5 Conversion ................................ 28
3.4.6 Preparing package and files .................................................. 28
3.4.7 Preparing CMakeLists.txt .................................................. 29
3.4.8 Code: node_braitenberg.h .................................................. 29
3.4.9 Code: node_braitenberg.cpp ................................................. 31
3.4.10 Code: braitenberg.cpp ...................................................... 36
3.4.11 Make .................................................................................. 38
3.4.12 Experiments ........................................................................ 38
3.5 Running ROS application from the server .................................. 40
3.6 Running ROS application remotely .......................................... 41

4 Conclusion .................................................................................. 42
List of Figures

2.1 Registration button is placed at the top of web page. 8
2.2 Fields to fill in for registration. 9
2.3 Login button on the SyRoTek web site. 10
2.4 Change password button is placed at the top of web page. 10
2.5 Change password form. 10
2.6 Password retrieval. 11
2.7 Place of courses section at the top of web. 11
2.8 The Enroll button in the Courses section. 12
2.9 Course description. 13
2.10 Task description with specification of required robots and their interfaces. 14
2.11 Reservation button under the course button in courses section. 15
2.12 Reservation duration button. 16
2.13 Available reservation selection. 17
2.14 Active reservation. 18
2.15 The Invite form. 19
2.16 Arena Control Panel. 20
2.17 Arena Control Panel - Robots, Docks. 21
2.18 Arena Control Panel – Obstacles, Session. 21
3.1 Braitenberg vehicle in Stageros simulator 39
3.2 Braitenberg vehicle in SyRoTek 40
Chapter 1

Introduction

This document is framed as a user manual, which will be utilized as a guideline for new users of the SyRoTek system. The SyRoTek ("System for RoboTic E-learning") allows you to remotely (via internet) control a multi-robot platform in a dynamic environment. With SyRoTek you will be able to develop your own algorithms and monitor their behaviour on-line during real experiments. You can train and test your robotic skills with a large set of pre-prepared single and multi-robot exercises and with comprehensive robotic courses you can enroll for. Besides, you can arrange your own multi-robot scenarios simply by a remote control of dynamic obstacles in the SyRoTek arena and by specifying required robots. SyRoTek provides a large set of sensors some placed onboard the robots, while others stand-alone to get global overview of the playfield status on-line. In this manual, the process from registering onto the system, enrolment of the first course and the first task therein to implementing your own application, will be described step by step.
Chapter 2

Web interface

2.1 Registration

The SyRoTek platform is world-wide available platform for tele-robotic education and research. Interested persons and institutions from the whole world are hearty welcome to use the system, but due to limited capacity, the access to SyRoTek is provided individually based on communication with system administrators. Although many functionalities are available for non-registered users (e.g., information about the system, documentation, demonstration videos, live view from the SyRoTek Arena, etc.), only a registered user can enroll courses and control robots (which is probably the reason why you want to use the system). First step of using the SyRoTek system is therefore registration. You are requested to fill the registration form at the SyRoTek main page https://syrotek.felk.cvut.cz/. You can do it by clicking on the Registration button, which is placed in the menu at the top of the web page (see Fig. 2.1).

You are requested to fill in the fields in the registration form (see Fig. 2.2) with your first name, last name, user name, e-mail address, organization and the reason why you want to use the SyRoTek system. Please fill the information carefully and truthfully (especially the reason field) as your request will be processed and reviewed personally by the SyRoTek administrator. Incomplete information may be a reason to not allow you to enter the system. Typically, the review process takes one or two working days after which you will receive acceptance message with login information onto the e-mail address specified in the registration form.

Figure 2.1: Registration button is placed at the top of web page.
2.2 Login

If you are successfully registered on the SyRoTek system, you can log in on the server by clicking the Login button at the top right corner of the SyRoTek web site (see Fig. 2.3), filling in the login fields and confirming then. This opens you possibilities of enrolling courses, make reservations of robots, controlling them and other functionalities. You can recognize that you are logged in at the top right corner.
2.3 Change the password

The password you obtained in the registration e-mail is really hard to remember. It is therefore highly recommended to change it. This can be done by clicking the *change password* button at the top right corner of the web (see Fig. 2.4). Filling the form (see Fig. 2.5) which appears is straightforward. Just enter your old password and then twice your preferred new one.

![Change password button](image)

Figure 2.4: *Change password* button is placed at the top of web page.

![Change password form](image)

Figure 2.5: Change password form.

2.4 Password retrieval

If you lose your password, you can retrieve it by filling the *Password retrieval* form. This form is placed below the *Login form*, see Fig. 2.6. Just fill in your e-mail address that you used for registration and the e-mail with a new password will be sent immediately to this e-mail.
CHAPTER 2. WEB INTERFACE

Login

![Login form](image)

Password recovery

If you forgot your password or if you are logging in for the first time, enter your e-mail into form below. You will receive recovery link, which allows you to set a new password.

![Password recovery form](image)

Figure 2.6: Password retrieval

2.5 Course enrollment

When you click on the Courses section, which is at the top of the web page, you can enroll for courses (see Fig. 2.7, 2.8). This is possible only if you are logged in. Enrolling a course is the first step to start controlling robots.

![Course enrollment interface](image)

Figure 2.7: Place of courses section at the top of web.

2.6 Task selection

The second step is to select the task you want to work on. In the Courses section, select the course containing your tasks. After that you can see the available tasks of the course, if any. After clicking the task, task description appears, which contains information about the task, specification of robot(s) to be used in the task and their interfaces (see Fig. 2.10).

![Task selection interface](image)
2.7 Reservation

The reservation system takes the advantage of the architecture of SyRoTek environment based on users’ tasks. As users are allowed to solve exclusively tasks defined by teachers within a course, also the reservation must be assigned to a particular task. Nevertheless, such a restriction helps users to specify their requirements on the robotic platform easily from the beginning. The required platforms and equipment for each task are defined by the teacher. Using this approach, users can reserve the system for solving all unique tasks already at the beginning of the course, when the knowledge of users on the robotic tasks is limited. More experienced users requiring to solve individually defined robotic experiments are required to ask administrators for a user status enabling to create new tasks.

Make a reservation button will be under each task in the page of a specific course as soon as you have enrolled for this course (see Fig. 2.11). The same button is located under the task title when you select the task (see Fig. 2.10).

Courses

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMR</td>
<td>Demos and tasks of IMR members</td>
<td>25</td>
</tr>
<tr>
<td>IMR_RESEARCH</td>
<td>IMR research projects</td>
<td>0</td>
</tr>
<tr>
<td>ECI</td>
<td>Introduction to mobile robotics 2011</td>
<td>77</td>
</tr>
<tr>
<td>MKR</td>
<td>Mobilní a kolektivní robotika 2011</td>
<td>5</td>
</tr>
<tr>
<td>MKR_12</td>
<td>Mobilní a kolektivní robotika 2012</td>
<td>21</td>
</tr>
<tr>
<td>MKR_13</td>
<td>Mobilní a kolektivní robotika 2013</td>
<td>16</td>
</tr>
<tr>
<td>MOR</td>
<td>Mobilní robotika 2010</td>
<td>8</td>
</tr>
<tr>
<td>PAR</td>
<td>Praktická robotika 2011</td>
<td>24</td>
</tr>
<tr>
<td>PAR_12</td>
<td>Praktická robotika 2012</td>
<td>26</td>
</tr>
<tr>
<td>PAR_13</td>
<td>Praktická robotika 2013</td>
<td>16</td>
</tr>
</tbody>
</table>

Create new course

Figure 2.8: The Enroll button in the Courses section.

Note that each robot has specified a symbolic name (typically one two, etc.) which is visible in its description. As you will see later, this name is used to control the robot.
When you click on the *Make reservation* button you will see the timetable of the SyRoTek reservation system. On the top of the page you can select a various durations and days of reservation (see Fig. 2.12). Under the calendar you can see the timetable of the SyRoTek system (similar to Fig. 2.13). Reservation is created by clicking the green text *reserve* at the preferred time and date. This makes reservation slot for you at the time and date you
2.7. RESERVATION

ECI_LABS: Labs

← Course: Introduction to mobile robotics 2011 | Reservations

Edit | Inherit

Motivation

Would you like to practice what you learnt during the lectures? Would you like to navigate a simulated/real robot? It is your time now!

Specification

The aim of this lab to let you play with a robot in the simulated environment and then with a real robot placed in SyRoTek arena. You can choose from the following tasks:

1. Wall following - the goal of the robot is to go to the obstacle and the follow it.
2. Obstacle avoidance - try to implement one of the Bug algorithms mentioned at Monday’s lecture.
3. Wandering - program the robot to randomly move in the environment without colliding with obstacles.

The robot is equipped with odometry and laser range-finder. To program the robot you should use the SyRoTek system.

Robots & Interfaces

Robot „one“:

Starting position is unspecified.
Interfaces:
- laser::ranger
- base::position2d

Figure 2.10: Task description with specification of required robots and their interfaces.
Praktická robotika 2013

Students and classification | Groups

Cílem kurzu je zprostředkovat studentům praktické dovednosti v oblasti řízení robotu ve složité úloze, tj. od řešení problémů návrhu architektury robotu, zpracování senzorických dat, přes navigaci a vytváření modelu prostředí až po plánování a inteligentní rozhodování. Důraz je kladen na cvičení, kde studenti budou řešit komplexní úlohu na reálném hardware mobilního robotu a budou mít dostatek prostoru pro implementaci a experimenty tak, aby bylo zřejmé, proč základní algoritmy ne vždy fungují a proč je vhodné použít sofistikovanější metody.

Exercises

1. task: Seznámení se systémem SyRoTek (PAR_13_INTRO)
   - Classification: Not satisfied. – Make a reservation
2. task: Plánování na binární mřížce (PAR_13_PLAN)
   - Classification: Not satisfied. – Make a reservation
3. task: Prohledávání neznámého prostředí (PAR_13_EXP)
   - Classification: Not satisfied. – Make a reservation
4. task: Lokalizace (PAR_13_LOC)
   - Classification: Not satisfied. – Make a reservation

Figure 2.11: Reservation button under the course button in courses section.

selected with the defined duration. The robots according to the task specification will be prepared in the Arena for this time, date and duration and you can use (control) them. After your reservation finishes, the robots will be automatically navigated to the docks and you will lose possibility to control them.

Of course, you can make another reservation for another time slots (if these slots are available, i.e. not occupied by other users). Note that you cannot choose the specific robot you want to use. Instead, the system selects the robot according to its sensor equipment, state of its batteries and previous occupancy of the robot automatically.
2.8 Active reservation

When you have done a reservation you can see the list of your reservations under Calendar on the right side of the SyRoTek web site (see Fig. 2.14). This is only possible when you are logged in. When your reservation is active you can also invite other user(s) to your session (i.e. to allow them to use/control the robots you have reserved) or control the SyRoTek system in the Arena Control Panel.

These panels are located on the right side of the web page (see Fig. 2.14) and are described in the next sections. In the description of active reservation are the course name, the exercise name, the start time of reservation, owner’s name, the names of invited users and the list of robots which can be used for the exercise.

2.9 Invite a user

If you have an active reservation you can invite colleagues from your course by clicking the check box in front of their names and confirming by clicking Set permissions, see Fig. 2.15. After that, the invited users can control your robots. Of course, you can take
them the rights. This can be done in the same form by unchecking the corresponding check box.

### 2.10 Arena view

You can see the current state of the Arena on the schematic view/live view of the SyRoTek system on the main page or on the Arena Control Panel. There are three cameras which permanently record and stream the situation in the Arena. Each of them is placed at a different place (one is on the top, one is used for the front view and the last one for the rear view), which gives the user a nice and detailed overview of the situation in the Arena, which allows to control the robots at distant.

### 2.11 Arena Control Panel

The Arena Control Panel shown on Fig. 2.16 gives necessary information about the current state of the SyRoTek system and its key components. On the left hand side you
2.11. ARENA CONTROL PANEL

SyRoTek

Figure 2.14: Active reservation.

You can see the schematic view of the Arena, while the panel with live view is on the right hand side. You can select which camera you would like to use with the tabs above the live view. The schematic view shows the current position of robots and obstacles as simple drawings which allow to see the current positions of the robots and obstacles even on slow networks. Please see the webpage footnotes for browser configuration instructions.

There are two types of obstacles in the arena. Some can be controlled by the obstacles manager and others can’t because they are fixed. Those which are fixed have a grey color on the schema. The red and white obstacles can be controlled. The red color means that the obstacle is ejected while the white means that the obstacle is retracted.

The control panel is under the schematic view. Using the list box you can select an obstacle or a robot (or by clicking it in the view). By using the arrows you can change state of the selected object (send a robot to the specified position and eject or retract obstacles). Robots can be driven to a specific location by clicking on the "go to" button or by right clicking the destination in the arena view. Robots can be also sent to the dock.

The tables giving an overview of the current state of the particular components of the SyRoTek system are placed next to the Control Panel. Fig. 2.16 shows an example of the table of active services. All statuses must be OK for correct arena functioning. Fig. 2.17
shows an example of the table of robot statuses that is shown when you click on the “Robots” tab. For each robot, the table gives its position (X, Y, heading) along with its current assignment, assigned dock, battery state, strength of wifi signal and description of failures if there are any. The next tab, Fig. 2.17, displays the docks state, which shows which docks are currently free or the occupied by some robot which is recharging.

Fig. 2.18 shows an example of table showing the state of controllable obstacles (tab
2.11. ARENA CONTROL PANEL

Figure 2.16: Arena Control Panel.

*Obstacles* and an example of table showing informations about your session (tab *Session*). In the *Arena* tab are information about the general arena state.
Figure 2.17: Arena Control Panel - Robots, Docks.

Figure 2.18: Arena Control Panel – Obstacles, Session.
Chapter 3

Command Line Interface

While beginners and novices to robotics and SyRoTek system prefer to command robots in an intuitive way of web interface, advanced users and users who develop theirs own robot control application use command line interface. To simplify and sweeten the work in command line and to write user applications, several tools are available on the SyRoTek server.

3.1 Connecting to the server

As all the functionalities are on the SyRoTek server, befor using the tools, you are required to connect to the server. The connection is set by ssh protocol and is started by th standard command

```
ssh user_name@syrotek.felk.cvut.cz
```

where user_name is your login. You are asked for your password and after you provide cor-
rect password, you are connected to the SyRoTek server (its IP is syrotek.felk.cvut.cz) and you can work with the system. Note, that you have to have an active reservation, if you want to test the commands and actions described in the following text (control robots and obstacles mainly).

3.2 syr_control

By command

```
/www/rb/syr_control.rb
```
it is possible to command the robot directly. The robot can be moved to the position 
\[ x, y, \phi \], rotated by the specified angle or moved forward by the specified distance. Note, 
that all angle are in radians and distances in meters. The format of the command is 
displayed when your run the command without parameters (see listing 3.1).

```
1 Robot Manager commander
2 usage:
3 syr_control g[oto] <id> <x> <y> <phi> — moves robot to the position \[ x, y, \phi \]
4 syr_control f[orward] <id> <l> — moves robot by l meters forward
5 syr_control r[otate] <id> <r> — rotates robot by r radians (positive 
6 value means counterclockwise direction)
7 syr_control o[bstacle] <id> <1/0> — raises (1) or retracts (0) movable
8 obstacle
```

Listing 3.1: Help for command /www/rb/syr_control.rb

### 3.3 ROS Launch file

The Robot Operating System (ROS) is used to communicate with the SyRoTek’s 
robots. As every robot has different sensors, for example some SyRoTek’s robots con-
tain sonar other laser range-scanner, you must to specify which sensor data should be 
provided for your application. You must enable all the sensors you want to use by ROS 
launch file. In folder /opt/syros/ are located two setting files named syros.launch and 
syros4.launch. Running one of these files you can start communication with the SyRoTek 
system. File named syros.launch is used for communication with one robot and it en-
ables messages from sonar and laser. Further it enables messages that give global position 
of robot and position of base (see listing 3.2).

```
1 <launch>
2   <!-- create a node named syros_position that is a type of
3       static_transform_publisher from tf package -->
4   <node pkg="tf" type="static_transform_publisher" name="syros_position"
5     args="0 0 0 0 0 0 syros/global_odom syros/ 100" />
6   <!-- create a node named syros that is a type of syros from syros
7       package -->
8   <node name="syros" pkg="syros" type="syros">
9     <!-- set a private parameters for the node
10        -->
11     <param name="syros/robacom_socket" value="/tmp/one" />
12     <!-- enable message that gives a position of base -->
13     <param name="syros/base/position2d/enable" value="true" type="bool" />
14     <!-- enable message that gives a data from laser -->
```

File named `syros4.launch` serves for communication with four robots. In this file you can see that messages can be remapped and that setting is used only to get global and base positions of robots (see listing 3.3).

```xml
<launch>
  <group ns="robot_0">
    <node pkg="tf" type="static_transform_publisher" name="robot_0_position" args="0 0 0 0 0 robot_0/global_odom syros/800" />
    <node name="syros" pkg="syros" type="syros">
      <param name="robacem_socket" value="/tmp/one" />
      <param name="driver_loop_period" value="500" />
      <param name="base/position2d(enable)" value="true" type="bool" />
      <remap from="syros/global_cmd_vel" to="/robot_0/cmd_vel"/>
      <remap from="syros/global_odom" to="/robot_0/odom" />
    </node>
  </group>

  <group ns="robot_3">
    <node pkg="tf" type="static_transform_publisher" name="robot_3_position" args="0 0 0 0 0 robot_0/global_odom syros/800" />
    <node name="syros" pkg="syros" type="syros">
      <remap from="syros/global_cmd_vel" to="/robot_0/cmd_vel"/>
      <remap from="syros/global_odom" to="/robot_0/odom" />
    </node>
  </group>
</launch>
```

Listing 3.2: syros.launch
CHAPTER 3. COMMAND LINE INTERFACE

These files are started by command `roslaunch` file (for example for communication with four robots it is `roslaunch syros4.launch`).

To verify that the setting is correct and the communication is working properly, it is possible to show list of active messages by using command `rostopic list`. The list of messages should be same as the requested. When this list is empty, the communication is not running and it is necessary to restart command on the server. When the list of active messages is still empty than contact the teacher.

3.4 Simple ROS program - Braitenberg vehicle

3.4.1 Introduction

This section aims to show how to write a simple application (simulation of Braitenberg vehicle) in the ROS to control the robot. At the beginning you will find an explanation of the algorithm that simulates the behavior of a Braitenberg vehicle. This will be followed by the instructions how to create a package and write program. Finally you will see results of experiments on a real robot.

Before you proceed, make sure that you meet the following requirements:

- Basic knowledge of programming in C++
- Installed ROS Groovy
- Prepared catkin workspace
- Basic knowledge of ROS - Beginner level tutorials (http://wiki.ros.org/ROS/Tutorials)
- Access to SyRoTek or Stageros simulator
3.4.2 Braitenberg Vehicle

The Braitenberg vehicle is an autonomous vehicle. It usually has two primitive sensors, which are directly connected to the motors. It means, that if right sensor sends higher values, right motor runs faster and robot turns left. By this simple behavior can be achieved an obstacle avoidance behavior.

3.4.3 Algorithm

Assume, you have robot with only laser scanner, which measures distances from the robot to obstacles. To simulate behavior of the Braitenberg vehicle, the algorithm finds the minimal distances on the left and right side (the closest obstacles). You can not simply set up left and right motor to follow these values as ROS allows to set up linear velocity and angular velocity, so there some conversion is needed (see bellow).

3.4.4 Processing inputs and outputs

Vehicle behavior is implemented in the class NodeBraitenberg2, which is realized as a ROS node. This class has two methods (except constructor and destructor). The first method processes laser scan data and the second controls robot.

Data from laser scan are in message type LaserScan. You can find out more information about this type of message by using command:

```
rosmcg show LaserScan
```

The result is:

```
[ sensor_msgs/LaserScan ]:
  std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
  float32 angle_min
  float32 angle_max
  float32 angle_increment
  float32 time_increment
  float32 scan_time
  float32 range_min
  float32 range_max
  float32 [ ] ranges
  float32 [ ] intensities
```

Listing 3.4: LaserScan message description
Vector `float32[] ranges` contains raw data from the laser scanner. As mentioned above, you can simulate Braitenberg vehicle by finding the minimal distances on the left and right side. Assuming that the sensor is positioned symmetrically, you can find minimum value from the first and second half of the `ranges` vector and use those values as the minimum on the left and right side. Knowing angles corresponding to the minimum values might be useful. If you know the index of the minimal value in the `ranges` vector, you can calculate the angle by using this equation:

\[
\phi = \left( i - \frac{l}{2} \right) \cdot \theta
\]  

(3.1)

Where

\( \phi \) is angle of \( i \)-th element of a vector

\( i \) is index of element in the vector

\( l \) is length of the vector

\( \theta \) is an angle incrementation (`angle_increment`)

Commands for the robot are sent through the Twist messages. You can check description of this type of message in the same way as before:

```bash
rosmsg show Twist
```

The result is:

```
1 [ geometry_msgs/Twist ]:
2 geometry_msgs/Vector3 linear
3 float64 x
4 float64 y
5 float64 z
6 geometry_msgs/Vector3 angular
7 float64 x
8 float64 y
9 float64 z
```

Listing 3.5: Twist message description

The Twist message contains 3 components for linear speed and 3 components for angular speed, but robot moves only in 2 dimensional space so \( x \) in linear (for linear velocity) and \( z \) in angular (for angular velocity) will suffice. You can leave other values equal to zero.
3.4.5 Conversion

Assuming that you have the minimum values, the only problematic part is to make conversion of these values to linear and angular velocity. Because the linear velocity has no effect (theoretically) on the direction of movement, it can be set to constant value. You can use following equation to calculate angular velocity:

\[ |\omega| = c \cdot \frac{d_{long}}{d_{short}} - c \]  (3.2)

Where

- \( |\omega| \) is absolute value of the angular velocity
- \( c \) is constant coefficient representing sensitivity to the ratio of the minimal distances
- \( d_{long} \) is greater value from minimal distances on left and right side
- \( d_{short} \) is smaller value from minimal distances on left and right side

As you can see, this formula provides us only with absolute value of the angular velocity. That will be sufficient, because sign determined by simple if/else construction (see below in code explanation).

You can now process input data, make conversion to linear and angular velocity and send commands to robot.

3.4.6 Preparing package and files

To create package in catkin, open your catkin workspace first. Then move to src/ directory. Following command will create catkin package:

\texttt{catkin_create_pkg dem_braitenberg roscpp rospy std_msgs}

This command creates a new directory named \texttt{dem_braitenberg} in \texttt{src/}. As a dependence on the package \texttt{roscpp} is included, there are 2 files in the new directory (\texttt{package.xml}, \texttt{CMakeLists.txt}) and 2 directories (\texttt{src/}, \texttt{include/}).

In \texttt{package.xml} you can fill information about package (author, version, name, description, ...).

Now, you need to create a class, which will operate robot, so create files:

- \texttt{include/node_braitenberg.h}
- \texttt{src/node_braitenberg.cpp}

You will also need main function, so there must be another file:

- \texttt{src/braitenberg.cpp}. 

CTU FEL, Intelligent and Mobile Robotics Group
3.4.7 Preparing CMakeLists.txt

In order to run program, you need to set up CMakeLists.txt. In this case, it will be only few lines at the end. First you must tell, which files needs to be compiled. You can create variable (in this case SRCS1), which contains all necessary source files. Add following lines to the end of CMakeLists.txt:

```
set (SRCS1 src/node_braitenberg.cpp)
set (SRCS1 src/braitenberg.cpp)
```

Listing 3.6: Set the source files in the CMakeList.txt

Next include catkin directories by following line:

```
include_directories(include ${catkin_INCLUDE_DIRS})
```

Listing 3.7: Include catkin directories in the CMakeLists.txt

Now create executable and link catkin libraries to it.

```
add_executable(braitenberg ${SRCS1})
target_link_libraries(braitenberg ${catkin_LIBRARIES})
```

Listing 3.8: Define output executable file and libraries to link

Everything is now prepared and you can start programming.

3.4.8 Code: node_braitenberg.h

```
#define SR_NODE_BRAINTENBERG2
#include "ros/rosc.h"
#include "sensor_msgs/LaserScan.h"
#include "geometry_msgs/Twist.h"

class NodeBraitenberg2
{
  public:
    /* Construction: */
  
  /* Demonstration task: "Braitenberg Vehicle"
  * This class controls robot. It's behavior is inspired by Braitenberg's
  * vehicle. In this case robot find minimal value on the left and right
  * side and goes, where the value is higher.
  * /
```

CTU FEL, Intelligent and Mobile Robotics Group
3.4. SIMPLE ROS PROGRAM - BRAITENBERG VEHICLE

```cpp
NodeBraitenberg2(ros::Publisher pub, double angleC, double speed);

private:
    void messageCallback(const sensor_msgs::LaserScan::ConstPtr& msg);

    void publishMessage();

    double angleCoef; // Coefficient for transferring angles to speed.
    double robotSpeed; // Speed of robot [m/s].
    double angleMinLeft; // Angle, at which was measured the shortest
distance on the left.
    double distMinLeft; // Minimum distance measured by sensor on the
left.
    double angleMinRight; // Angle, at which was measured the shortest
distance on the right.
    double distMinRight; // Minimum distance measured by sensor on the
right.
ros::Publisher pubMessage; // Object for publishing messages.
```

Listing 3.9: header file node_braitenberg.h
CHAPTER 3. COMMAND LINE INTERFACE

Code explanation

4 #include "ros/ros.h"
5 #include "sensor_msgs/LaserScan.h"
6 #include "geometry_msgs/Twist.h"

Listing 3.10: Include required headers

The header file ros.h is necessary to include. It covers all the headers necessary to use the most common pieces of the ROS system. In order to be able to send and receive messages, appropriate headers must be included for each type of message, that are used in program (in this case: LaserScan and Twist).

26 NodeBraitenberg2(ros::Publisher pub, double angleC, double speed);

Listing 3.11: Constructor

Constructor creates the instance of the class and saves input values to appropriate variables.

39 void messageCallback(const sensor_msgs::LaserScan::ConstPtr& msg);

Listing 3.12: Callback method

The messageCallback method is called by the ROS node every time, when a new data of the LaserScan are published. This method processes data from laser scanner (msg), finds the minimum values, saves them to appropriate internal variables and call method publishing commands for the robot (see bellow).

50 void publishMessage();

Listing 3.13: Publish method

The publishMessage method sends the message Twist with the required speeds. It reads data stored in the class internal variables (distMinLeft, distMinRight), makes conversion to the velocities (Equation 3.2) and publishes them as a Twist message, that is processed by the robot.

54 double angleCoef; // Coefficient for transferring angles to speed.

Listing 3.14: angleCoef

This variable is constant coefficient representing the speed sensitivity to the ratio of the minimal distances (\(c\) in equation 3.2).

3.4.9 Code: node_braitenberg.cpp
3.4. SIMPLE ROS PROGRAM - BRAITENBERG VEHICLE

```c++
#include "node_braitenberg.h"

// Constructor and destructor
NodeBraitenberg2::NodeBraitenberg2(ros::Publisher pub, double angleC, double speed)
{
    angleCoef = angleC;
    robotSpeed = speed;
    pubMessage = pub;
}

NodeBraitenberg2::~NodeBraitenberg2()
{
}

// Publisher
void NodeBraitenberg2::publishMessage()
{
    // preparing message
    geometry_msgs::Twist msg;
    if (distMinLeft >= distMinRight)
    {
        msg.angular.z = angleCoef*distMinLeft/distMinRight - angleCoef;
    } else
    {
        msg.angular.z = -(angleCoef*distMinRight/distMinLeft - angleCoef);
    }
    msg.linear.x = robotSpeed;
    if (distMinLeft < 0.25 && distMinRight < 0.25 && angleMinLeft < 0.7 && angleMinRight < 0.7)
    {
        msg.angular.z *= 50;
        msg.linear.x *= 0.5;
    }
    // publishing message
    pubMessage.publish(msg);
}

// Subscriber
void NodeBraitenberg2::messageCallback(const sensor_msgs::LaserScan::ConstPtr& msg)
{
    // Calculation of array size from angle range and angle increment.
    int size = msg->ranges.size();
    int minIndexLeft = 0;
    int minIndexRight = size / 2;
    // This cycle goes through array and finds minimum on the left and right
```
for(int i=0; i<size/2; i++)
{
    if (msg->ranges[i] < msg->ranges[minIndexRight] && msg->ranges[i] > 0.05){
        minIndexRight = i;
    }
}
for (int i = size/2; i <size ; i++)
{
    if (msg->ranges[i] < msg->ranges[minIndexLeft] && msg->ranges[i] > 0.05){
        minIndexLeft = i;
    }
}

// Calculation of angle from indexes and storing data to class variables
angleMinLeft = (minIndexLeft-size/2)*msg->angle_increment;
distMinLeft = msg->ranges[minIndexLeft];
angleMinRight = (minIndexRight-size/2)*msg->angle_increment;
distMinRight = msg->ranges[minIndexRight];

// Invoking method for publishing message
publishMessage();

Listing 3.15: Source file node_braitenberg.cpp

**Code explanation - constructor**

NodeBraitenberg2::NodeBraitenberg2(ros::Publisher pub, double angleC, double speed)
{
    angleCoef = angleC;
    robotSpeed = speed;
    pubMessage = pub;
}

Listing 3.16: Test

As mentioned before, constructor creates new instance of the class and saves the input data to appropriate internal variables.

**Code explanation - publishMessage**

The `publishMessage` method creates and send messages with velocity commands for the robot.
3.4. SIMPLE ROS PROGRAM - BRAITENBERG VEHICLE

```cpp
19  geometry_msgs::Twist msg;
```

Listing 3.17: Twist message variable

First you need to prepare an empty message.

```cpp
21  if (distMinLeft >= distMinRight)
22  {
23      msg.angular.z = angleCoef*distMinLeft/distMinRight-angleCoef;
24  }
25  else
26  {
27      msg.angular.z = -(angleCoef*distMinRight/distMinLeft-angleCoef);
28  }
```

Listing 3.18: Angular velocity conversion

This code converts the minimal distances to required angular velocity of the robot. This conversion follows the equation 3.2. Result of the equation is absolute value of angular velocity. Sign of angular velocity is positive, if obstacle on the left is farther than obstacle on the right (robot will turn left), otherwise it is negative.

```cpp
30  msg.linear.x = robotSpeed;
```

Listing 3.19: Linear velocity

The linear velocity is set to the constant value defined in the constructor.

```cpp
31  if (distMinLeft < 0.25 && distMinRight < 0.25 && angleMinLeft < 0.7 && angleMinRight < 0.7)
32  {
33      msg.angular.z *=50;
34      msg.linear.x *=0.5;
35  }
```

Listing 3.20: Velocity modification

This part of code is not required for basic function of Braitenberg vehicle, however it greatly increases the likelihood of avoiding obstacles. If robot’s direction is close to perpendicular to the wall, difference between `distMinLeft` and `distMinRight` is small. The angular velocity will be also small and robot might not be able to avoid the wall in time.

This part of the code checks, whether there is obstacle in front of robot and if yes, it will lower linear velocity and greatly increase angular velocity (usually up to maximum). Robot will avoid the wall and continue according to original Braitenberg vehicle algorithm.

```cpp
38  pubMessage.publish(msg);
```

Listing 3.21: Publishing the Twist message

The publisher initialized in the class constructor sends the prepared message to the robot.

CTU FEL, Intelligent and Mobile Robotics Group
Code explanation - messageCallback

The `messageCallback` method is called by ROS system, when a new `LaserScan` message is available and this message is passed as a method parameter. The only input necessary for the Braitenberg vehicle algorithm are distances to the nearest obstacles on the left and right side of the robot. These distances are calculated from the vector `float32[] ranges (27)`, which contains raw data from laser range scanner.

```c
int size = msg->ranges.size();
int minIndexLeft = 0;
int minIndexRight = size/2;
```

Listing 3.22: Splitting the ranges vector

At first, the vector is split to two parts of equal size representing the left and right sides of the laser scan. It is assumed, that the laser scanner is placed symmetrically on the robot. Variable `minIndexLeft` contains index of minimum value on the left side and `minIndexRight` contains index of minimum value on the right side. Initial values of these variables are indexes, where `for` cycles begin.

```c
for(int i=0; i<size/2; i++)
{
    if (msg->ranges[i] < msg->ranges[minIndexRight] && msg->ranges[i] > 0.05)
    {
        minIndexRight = i;
    }
}
```

Listing 3.23: Finding the minimum

The `for` cycle is used to find index of minimum value on the left side (indexes from 0 to `size/2`). The condition `(msg->ranges[i] > 0.05)` is necessary for filtering data where occasional laser scanner errors are represented as zero reading in the raw data. This condition avoids finding these error codes as minimal values.

```c
angleMinLeft = (minIndexLeft-size/2)*msg->angle_increment;
distMinLeft = msg->ranges[minIndexLeft];
angleMinRight = (minIndexRight-size/2)*msg->angle_increment;
distMinRight = msg->ranges[minIndexRight];
```

Listing 3.24: Minimal values stored in the internal variables

The minimal values and respective indexes are found and saved to appropriate internal class variables. Geometrical angles to the nearest obstacles (on the left and right) are calculated from position in the range vector according to equation 3.1.

```c
publishMessage();
```

Listing 3.25: Calling the publishMessage method

Finally, the method `publishMessage()` described above is called.
3.4.10 Code: braitenberg.cpp

```cpp
#include "node_braitenberg.h"

#define SUBSCRIBER_BUFFER_SIZE 1  // Size of buffer for subscriber.
#define PUBLISHER_BUFFER_SIZE 1000 // Size of buffer for publisher.
#define ANGLE_COEF 2  // Coefficient for angles.
#define ROBOT_SPEED 0.10 // Speed of robot [m/s].
#define PUBLISHER_TOPIC "/syros/base_cmd_vel"
#define SUBSCRIBER_TOPIC "/syros/laser_laser"
#define SUBSCRIBER_TOPIC "/base_scan"

/* task: "Braitenberg vehicle 2"
 * In main function is created Subscribing node, which transmits messages
 * to NodeBraitenberg2 object. There are the messages processed and
 * generated.
 */

int main(int argc, char **argv)
{
    // Initialization of node
    ros::init(argc, argv, "braitenberg2");
    ros::NodeHandle n;

    // Creating publisher
    ros::Publisher pubMessage = n.advertise<geometry_msgs::Twist>(
        PUBLISHER_TOPIC, PUBLISHER_BUFFER_SIZE);

    // Creating object, which stores data from sensors and has methods for
    // publishing and subscribing
    NodeBraitenberg2 *nodeBraitenberg2 = new NodeBraitenberg2(pubMessage,
        ANGLE_COEF, ROBOT_SPEED);

    // Creating subscriber
    ros::Subscriber sub = n.subscribe(SUBSCRIBER_TOPIC,
        SUBSCRIBER_BUFFER_SIZE, &NodeBraitenberg2::messageCallback,
        nodeBraitenberg2);
    ros::spin();

    return 0;
}
```

Listing 3.26: Source code file braitenberg.cpp

Code explanation
CHAPTER 3. COMMAND LINE INTERFACE

You need to set size of buffer for publisher and subscriber (size of subscriber buffer is only 1, because only latest data are important).

Listing 3.27: Buffer size definition

Listing 3.28: Coefficient definition

Default values of the coefficients are defined here. Higher value in \texttt{ANGLE\_COEF} (c in equation 3.2) will cause robot to use higher angular speed even though the ratio of minimal distances is not too big. Value in \texttt{ROBOT\_SPEED} represents maximal linear velocity. These values are used in constructor of \texttt{NodeBraitenberg2}.

Listing 3.29: Definition of topic default names

The default names of the subscribed and published topic are defined in this section. It is possible to change the topic names using the ROS mapping mechanism at the run time.

Commented topics are for using when the program is run on SyRoTek and uncommented ones are used for testing with Stage simulator.

Listing 3.30: Initialization

This part of code initializes ROS node. This step must be done before you can register any publisher or subscriber.

Listing 3.31: Registering the publisher of the Twist message

This line creates publisher for \texttt{Twist} messages (velocity commands for the robot). The topic name defined above as \texttt{PUBLISHER\_TOPIC} is used. This step is only a registration and so far this publisher does not publish any messages. That will happen after calling \texttt{publishMessage(33)} method in \texttt{NodeBraitenberg2}.
3.4. SIMPLE ROS PROGRAM - BRAITENBERG VEHICLE

```java
NodeBraitenberg2 nodeBraitenberg2 = new NodeBraitenberg2(pubMessage, ANGLE_COEF, ROBOT_SPEED);
```

Listing 3.32: Creating the instance of NodeBraitenberg2 class

The constructor of the `NodeBraitenberg2` is called and parameters are the publisher created above and defined values.

```java
ros::Subscriber sub = n.subscribe(SUBSCRIBER_TOPIC, SUBSCRIBER_BUFFER_SIZE, &NodeBraitenberg2::messageCallback, nodeBraitenberg2);
```

Listing 3.33: Subscribing to the LaserScan topic

Now, when instance of `NodeBraitenberg2` class is prepared, the topic `LaserScan` is subscribed and the callback method `messageCallback` is registered. The method receives messages from laser scanner from now.

```java
ros::spin();
```

Listing 3.34: Spin the ROS loop

This code enters a ROS loop and starts to process the message queue and calling message callbacks.

### 3.4.11 Make

Now, everything is prepared, so open your catkin workspace in terminal and call:

```
catkin_make
```

This command should compile all packages in your catkin workspace. Braitenberg vehicle can be run by using this command:

```
rosrun dem_braitenberg braitenberg
```

### 3.4.12 Experiments

Stageros simulator

Before running application on SyRoTek, you should try it on simulator first. If you have successfully installed stage simulator and have world file with SyRoTek arena, start simulator by command:
Figure 3.1: Braitenberg vehicle in Stageros simulator
3.5. RUNNING ROS APPLICATION FROM THE SERVER

When you run your application, the robot in simulator should start moving immediately. As you can see in Figure 3.1, Braitenberg vehicle is successfully keeping minimal distances on the left and right at close values (marked with orange lines). It can even go between two obstacles that are spaced close.

SyRoTek

In order to run this program on SyRoTek, you need to set up right publisher and subscriber. More information is in 3.4.10. From Fig. 3.2 you can see that behavior remains same as in Stage and robot is successfully simulating Braitenberg vehicle.

3.5 Running ROS application from the server

It is possible to run your application directly on the server when using the ssh connection. At first, you need to run the ROS master. It is done by calling the command roslaunch
as described in section 3.3. The ROS application needs to be compiled and linked with the command `catkin_make` see section 3.4.11. To run the ROS application use the command `rosrun package_name executable_file` an example can be seen also in section 3.4.11.

3.6 Running ROS application remotely

To run the application remotely, it is necessary to set the correct connection to the ROS master running on the SyRoTek server. When you are connected to the local network at Czech Technical University in Prague, Charles square (Karlov náměstí), it is possible to use a direct connection to the SyRoTek system by setting two commands

```bash
export ROS_MASTER_URI="http://syrotek.felk.cvut.cz:11311",
export ROS_IP=xxx.xxx.xxx.xxx,
```

where `xxx.xxx.xxx.xxx` is your IP address.

As the ROS master is not started automatically, it is necessary to run the ROS master on the server using the ssh connection and the `roslaunch` command. It runs the `roscore` and `syros` node.

To check that the communication with server works you can use command `rostopic list`. You can also command the robot using `rostopic pub command`:

```bash
rostopic pub /syros/base_cmd_vel geometry_msgs/Twist 'linear: x: 0.01 '
```

Note that before trying the last command, `roscore` and `syros` node have to be running.

In the case of any problem, it is necessary to check, if the connection can be established. You can try to ping the SyRoTek server from your computer, and also is necessary try to ping your computer from the server. Another source of problem can be in the network name resolution. It can the help to use the numeric network address.
Chapter 4

Conclusion

As you read and follow the instruction through the whole guide, you are now registered on the SyRoTek system, and you are able to create the ROS node, which controls the robot on the SyRoTek arena.

The aim of this guide is to give you a tour of SyRoTek system from the very first steps (registration) to the first application on the real robots. This guide points out the most common problems and provides the usual solutions. If you still have any unsolved problems, do not hesitate to contact the SyRoTek tutors. This guide is not able to solve all the possible problems and also has no ambition to do so. Even the SyRoTek is a distant learning system, we encourage you to contact the human teachers in case of any questions or problems.

The effort of creating this unique robotic distant learning system and all relevant materials was or is supported by the Ministry of Education of the Czech Republic under Projects No. 2C06005, 7AMB12AR022 and FRVS872/2013, and the Ministry of Science of Argentina under Project No. ARC/11/11.