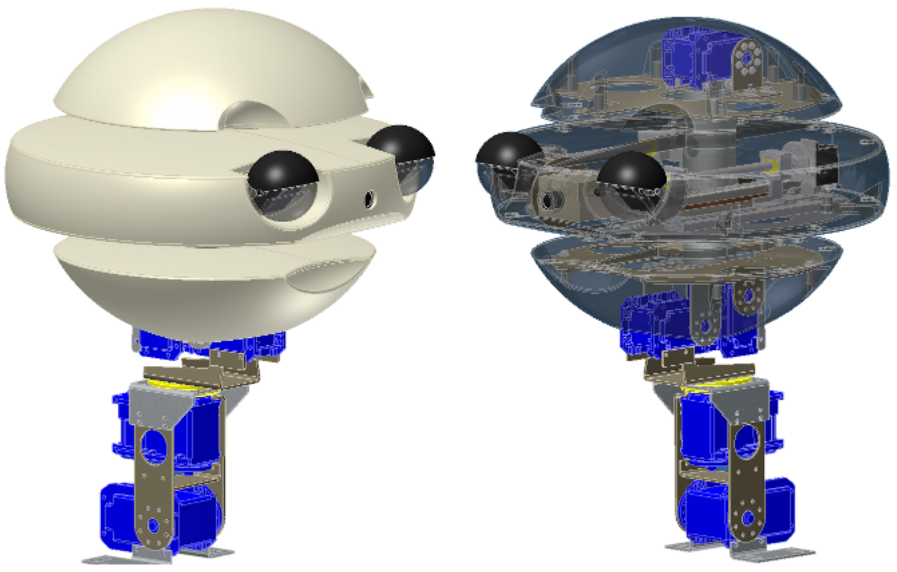
Emys User Guide



Wrocław University of Technology

CAD Mech sp. z o.o.

Kubasek Krzysztof Design Studio

2010

Emys User Guide
Copyright © 2010 by Wroclaw University of Technology

Lead Editor: Lukasz Malek
Editors Board: Jan Kedzierski, Adam Oleksy
Reviewer: Robert Muszynski

Wybrzeze Wyspianskiego 27, 50-370, Wroclaw, Poland

Safety

Do **NOT** power on the robot before reading and fully understanding the operation procedures explained in this manual. Neither the robot, nor the program is bug free, accidents can; you have to make sure that the robot always maintains a safe distance from people during operation.

The robot should be turn off (i.e. the power switch should be on the OFF position) when not in used.

Robot **MUST** be firmly attached to fixed basis.

Robot **MUST** be secured from hitting obstacles.

WARNING: In case of sudden power loss robot will be not controlled and will move freely what might lead to unexpected contact with human and/or obstacles.

WARNING: It is highly recommended **NOT** to change the range of the movements.

WARNING: Failure to follow these safety guidelines could cause serious injury or death to a user and/or damage to the robot.

Contents

1

Introduction

Emys is a robotic head designed to make long-term relationship with a human. It was initially prototyped at the Wroclaw University of The Technology in the European funded research project named LIREC . Unique shape of Emys was designed by Krzysztof Kubasek. The mechanical project and parts manufacturing was done by CadMech. Control system for Emys was created at Wroclaw University of Technology.

A head for socially aware robot plays a fundamental role in the interaction with humans. It should be able to generate gestures and emotions easily understandable for a human. This requires that the head should be in some way similar to the human one. Finding a good shape of the robot head is a very crucial problem.

Generally, it is very hard to create an anthropomorphic head that is well accepted by people. Therefore, Emys does not pretend to be similar to a human. It is technically inspired and its shape is in some way similar to turtle comics like head that explains the etymology of the Emys name (Emys orbicularis is latin name for European pond turtle).

Emys head consists of three separated discs what can be seen in Figure 1.1. The middle disc is mounted stiffly to a neck while the lower and the upper ones are able to change their pitch angles. The head is mounted to a basis via the neck, which is able to change the pitch and the yaw angles. In Emys middle disc two eyes are mounted, that are able to move out of it. Each eye has an eyelid which can rotate by changing pitch and roll angles. Please notice that Emys eyes do not allow to perceive. Emys is able to capture images with a camera mounted in the middle disc between the eyes. Due to the location of the camera it is usually interpreted by humans as Emys nose hole.

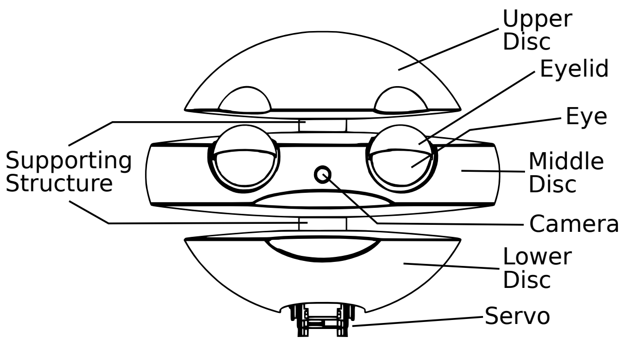


Figure 1.1: Emys head

1.1 Key features

- Height 35 cm (to head connector)
- 11 degrees of freedom (3 – neck, 1 – chin, 1 – forehead, 2 – eyes, 4 – eyelids)
- 6 digital servos (Robotis Dynamixel RX-28 and RX-64)
- 4 analog RC servos (Hitec HS-65HB)
- 2 motor driven slide potentiometer (Alps RS10N11M B103)
- Outer head shells manufactured in SD fast prototyping technology
- Freescale HC9S12A64 microcontroller
- EIA-485 standard for electrical characteristics of communication interface
- Robotis Dynamixel protocol
- Logitech QuickCam Sphere AF camera
  - Carl Zeiss® optics
  - Autofocus lens system
  - Ultra-high resolution 2-megapixel sensor with RightLight™
  - Color depth: 24-bit true color
  - Video capture: Up to 1600 by 1200 pixels (HD quality)
  - Frame rate: Up to 30 frames per second

  - High-Speed USB 2.0

- Logitech 4-Port USB Hub (2 ports available for the user)
- RS-232 – EIA-485 Converter
- Manhattan Serial-USB Converter
- Power supply single phase 100 – 240 VAC. (Fused 2.6 A)
- Windows and Linux compatible

1.2 Product Contents

After the opening of Emys box please ensure that it contains all the elements presented in Figure 1.2. If any of them is missing please contact your distributor.

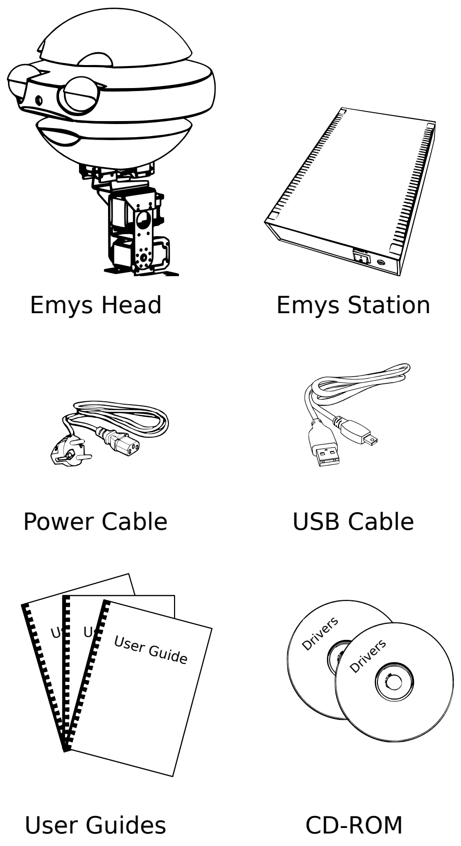


Figure 1.2: Emys box content

The package includes the following CD-ROMs:

- Emys Solution Disc,
- Manhattan Serial-USB Converter Disc,
- Logitech QuickCam Sphere AF Disc,

and documents:

- Emys User Guide,
- Emys Programmer Guide,
- Robotis Dynamixel RX-28 Manual,
- Robotis Dynamixel RX-64 Manual,
- Manhattan Serial-USB Converter User Guide,
- Logitech QuickCam Sphere AF User Guide,
- Logitech 4-Port USB Hub User Guide.

1.3 Requirements

To opperate Emys robot requires a PC computer with Windows XP/Vista/7 or Linux system with at least one USB 1.1 or 2.0 port free.

Before operating the head must be firmly attached to a fixed basis. Emys head must be secured from hitting hard objects.

2

Work with Emys

2.1 Emys Station

Emys Head is delivered with Emys Station. This part is not required by Emys Head to opperate, but it makes work with the robot much easier. The Station part integrates the following components:

- Power Supply,
- USB Hub,
- USB to EIA-485 Converter.

Emys Station has two functional panels, the one in the front of it and the other on its back. The front panel is equipped with a power switch and a LED indicating power status. The following sockets are placed on the rear panel of Emys Station: power, USB, Emys servos network. The elements layout on the panels is illustrated with Figures 2.1 and 2.2.

2.2 Setting up Emys

This section describes how to prepare Emys head to operate. To this aim one should:

1. Firmly attach Emys Head to a fix basis (e.g. to a table with two carpenter clamps or to a robot with screws).
2. Connect Emys servo network cable to Emys Station (Figure 2.3).

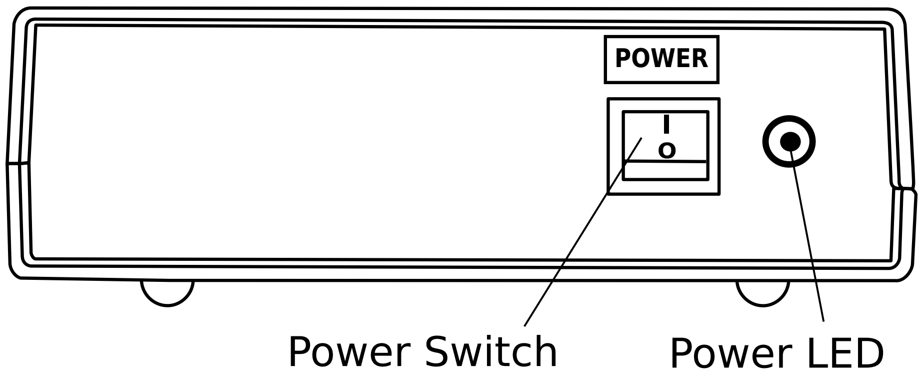


Figure 2.1: Front panel of Emys Station

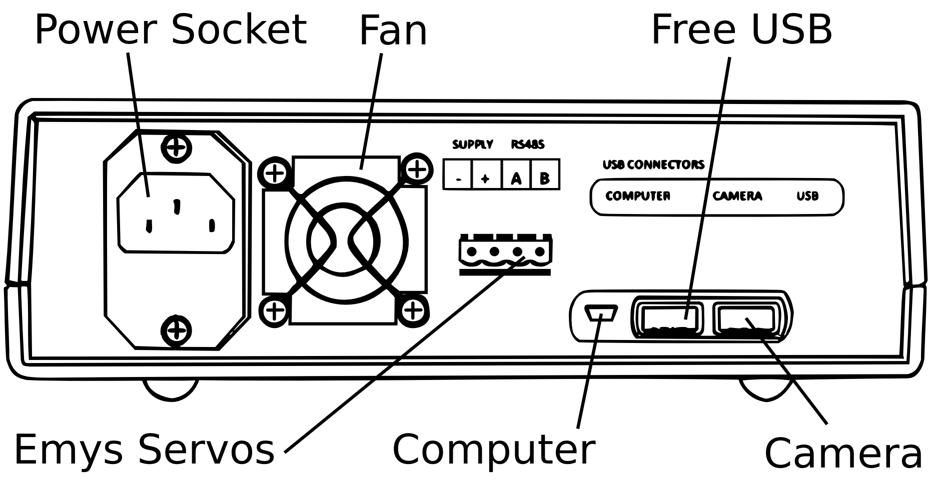


Figure 2.2: Rear panel of Emys Station

3. Connect Emys camera cable to Emys Station (Figure 2.3).
4. Install Manhattan Serial-USB Converter driver from the attached CD (Windows only).
5. Install Logitech QuickCam Sphere AF driver from attached CD. When asked to connect camera please connect Emys Station miniUSB socket with your PC USB socket.

2.3 Switching on Emys

After setting up Emys head one can switch it on. To this aim please

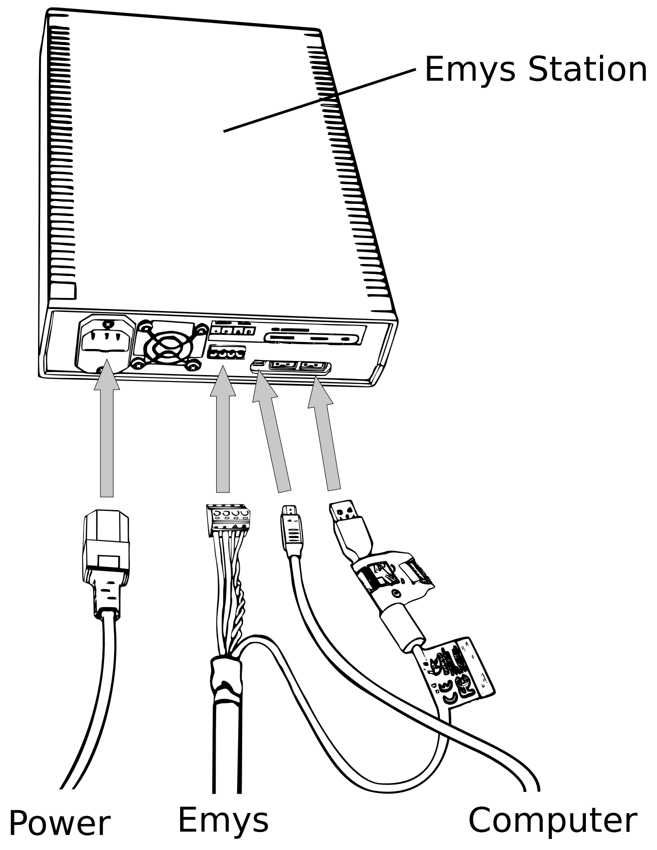


Figure 2.3: Connecting cables to Emys Station

1. Connect Emys Station to a single phase 100 – 240 VAC (Figure 2.3).
2. Grab gently Emys head middle disc and place it in the approximately vertical position.
3. Switch on Emys Station.
4. Release Emys head which is now ready to use.

2.4 Switching off Emys

This section describes how to finish work with Emys head.

1. Stop all Emys head servos.
2. Gently grab Emys head middle disc.

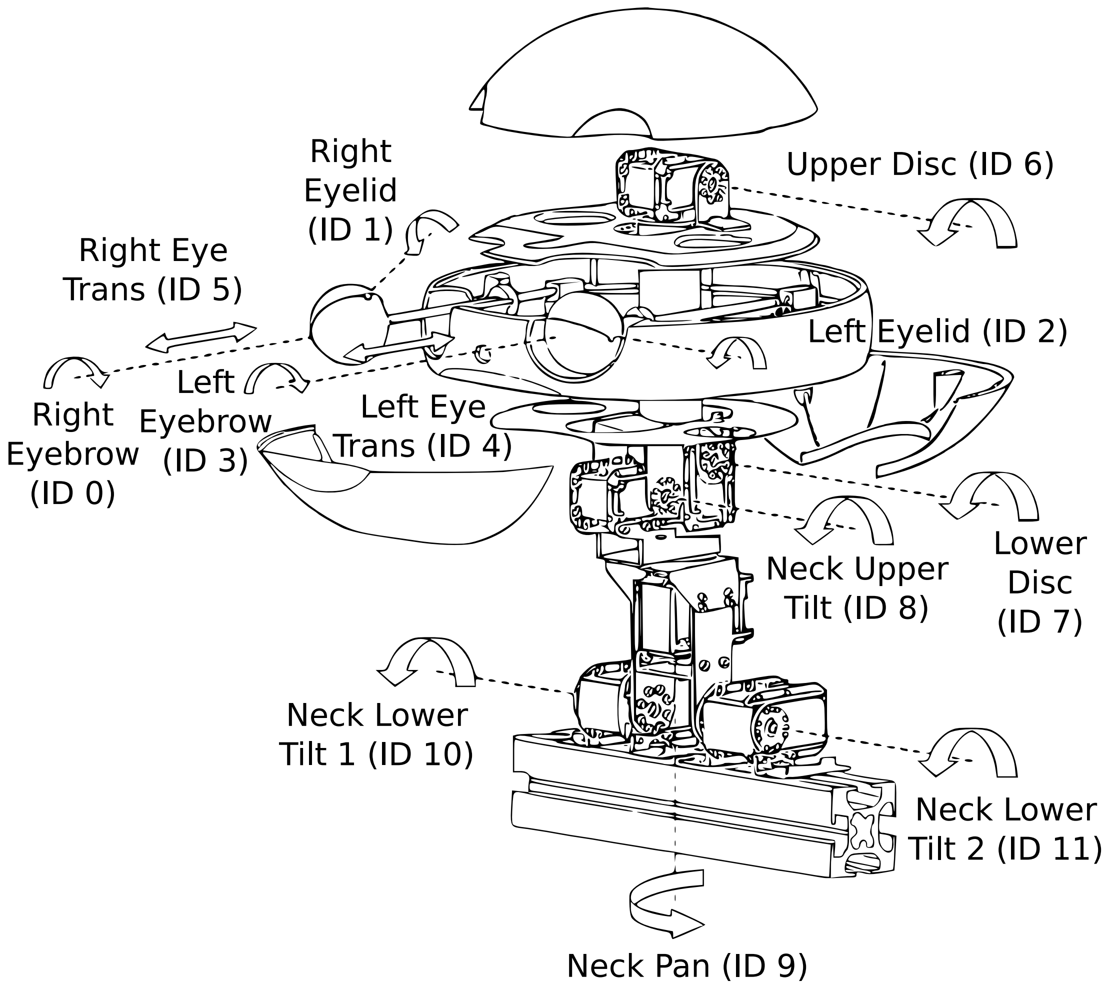


Figure 2.4: Emys joints layout

3. Turn off Emys Station.
4. Move gently Emys head to a secure position.
5. Disconnect Emys Station from power.
6. Disconnect Emys Station from PC.

2.5 Emys servos

Emys head contains 12 servos recognised by an individual number (ID), indicated below

ID 0 Right eyebrow (HS-65HB)

ID 1 Right eyelid (HS-65HB)

ID 2 Left eyelid (HS-65HB)

ID 3 Left eyebrow(HS-65HB)

ID 4 Left eye transition (RS10N11M B103)

ID 5 Right eye transition (RS10N11M B103)

ID 6 Upper disc (RX-28)

ID 7 Lower disc (RX-28)

ID 8 Upper neck (RX-28)

ID 9 Neck pan (RX-64)

ID 10 Lower neck 1 (RX-64)

ID 11 Lower neck 2 (RX-64)

Servos Placement in the head and the joints related with them are presented in Figure 2.4. All the servos are connected into one network based on EIA-485. The communication with all the servos happens via Robotis Dynamixel protocol (see Dynamixel manual).

INFO: Please note that not all the driving the eyes fully implement Dynamixel protocol.

The servo communication network cable consist of 4 wires. In Emys head these wires are marked according to the convention presented in Table 2.1. Emys network is

Emys symbol	Description
-	GND, ground, zero
+	VCC, power supply line
A	non-inverting data line
B	inverting data line

Table 2.1: Emys network lines

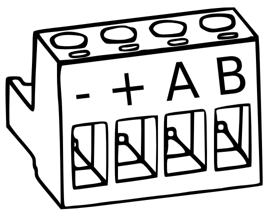


Figure 2.5: Emys network terminal connector

terminated by a 4 pin euro style connector presented in Figure 2.5.

INFO: All the servos mounted in the head have Status Return Level (address 16 [`0x10`]) set to 1. It means that, the servo returns data only for the `READ_DATA` command.

WARNING: It is highly recommended **NOT** to change the range of the servos movements.

2.6 Eye movement servos limited functionality

Servos driving the robot eyes, indicated by ID numbers from 0 to 5, do not fully implement Dynamixel protocol. Their functionality allows for obtaining the elements movements, but they do not support advanced Dynamixel servos features. For all these 6 servos one Freescale MC9S12A64 microcontroller is utilised to translate Dynamixel protocol commands to the individual analog servos commands.

Servos marked as ID 0 . . . 5 implement the following Dynamixel commands:

PING (`0x01`) No execution. It is used when controller is ready to receive Status Packet.

READ DATA (`0x02`) This command reads data from servos.

1. Parameters (start address must be 0 and number of parameters must be 19)
2. Position (address 36 [`0x24`], number of parameters must be 2)
3. Moving (address 46 [`0x2E`], number of parameters must be 1)

WRITE DATA (`0x03`) This command writes data to servos.

1. Enable/Disable torque (address 24 [`0x18`] , number of parameters must be 1)

2. Enable/Disable LED (address 25 [`0x19`], number of parameters must be 1)
3. Set Position (address 30 [`0x1E`], number of parameters must be 2)
4. Set Position and Velocity (address 30 [`0x1E`], number of parameters must be 4)
5. Set Velocity (address 32 [`0x20`], number of parameters must be 2)

2.7 Camera

The camera mounted inside Emys middle disc has been rotated along the roll angle by 90 degrees from its regular position. Therefore, the image from the camera has to be transformed by 90 degrees to the right. This can bee easily achieved with use of OpenCV library. The transformation in C/C++ language can be done with the following code:

Listing 2.1: Image rotation with OpenCV in C++

```
IplImage* frame1 = cvQueryFrame(capture);
IplImage* correct = CvCreateImage(CvSize(image->height,
                                         image->width),
                                  IPL_DEPTH_8U, 3);
CvFlip(image, image, -1);
CvTranspose( image, correct);
```

To perform the image transformation in C# language it is useful to apply `cvlib` – OpenCV wrapper.

Listing 2.2: Image rotation with OpenCV in C#

```
IplImage image = cvlib.CvQueryFrame(ref capture);
IplImage correct = cvlib.CvCreateImage(new CvSize(image.height,
                                                  image.width),
                                       (int)cvlib.IPL_DEPTH_8U,
                                       3);
cvlib.CvFlip(ref image, ref image, -1);
cvlib.CvTranspose(ref image, ref correct);
```

3

Software examples

Emys joints movements are realized by servos that are able to communicate via Dynamixel protocol. Therefore any software able to control such kind of servos will be able to control Emys. Dynamixel protocol is based on EIA-485 and is completely described in Dynamixel Servo Manual. To give a new users a quick start in application development, Emys is provided with working examples for Windows and Linux. Examples illustrate both how to communicate with single servo and how to perform series of actions on many servos concurrently.

3.1 Windows

For Windows environment five examples are provided. Two examples are written in C++ and three in C#.

3.1.1 `very_simple_EMYS`

Info: `very_simple_EMYS` is a command line application that shows how to use serial port for sending motion parameters to EMYS head.

Langauage: This example is delivered both in C++ and C#.

Compiler: Microsoft Visual Studio 2008

Usage:

```
>> very_simple_EMYS.exe COM ID position velocity
```

COM — name of the serial port

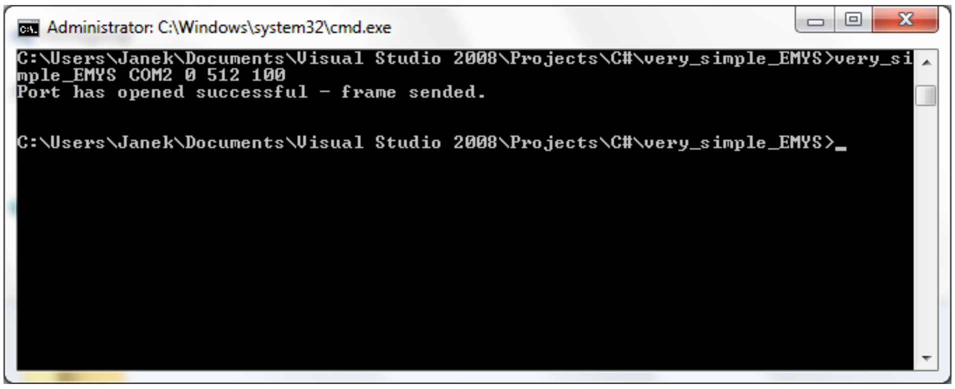
ID — identification number of the servo that will operate (see Figure 2.4)

position — goal position $[0, \ldots, 1023]$

velocity — maximum velocity during movement $[0, \ldots, 1023]$

Example:

```
>> very_simple_EMYS.exe COM1 0 512 100
```



```
C:\Users\Janek\Documents\Visual Studio 2008\Projects\C#\very_simple_EMYS>very_si
mple_EMYS COM2 0 512 100
Port has opened successful - frame sended.

C:\Users\Janek\Documents\Visual Studio 2008\Projects\C#\very_simple_EMYS>_
```

3.1.2 simple_EMYS

Info: simple_EMYS application allows for moving a single joint to a given position with given velocity. It is very similar to very_simple_EMYS, but it is extended by a graphical interface.

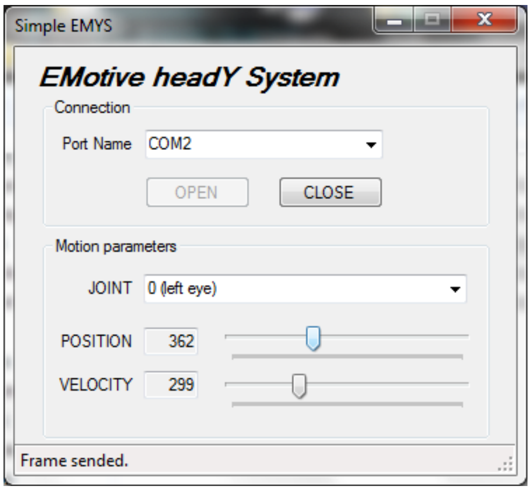
Language: This example is delivered both in C++ and C#.

Compiler: Microsoft Visual Studio 2008

Usage:

```
>> simple_EMYS.exe
```

Choose port name that Emys is connected to. Click OPEN button to open the port. Select desired joint from JOINT combobox (see Figure 2.4). Control joint position and velocity with sliders.



Port Name: COM2
JOINT: 0 (left eye)
POSITION: 362
VELOCITY: 299
Frame sended.

3.1.3 `EMYS`

Info: `EMYS` is the most complex example . This application allows for reproduction of movement series. It provides graphical interface that allows to set desired configuration of Emys joints. Operator can modify for each joint both position and velocity. Positions can be stored in memory and reproduced step by step.

Language: This example is developed in C#.

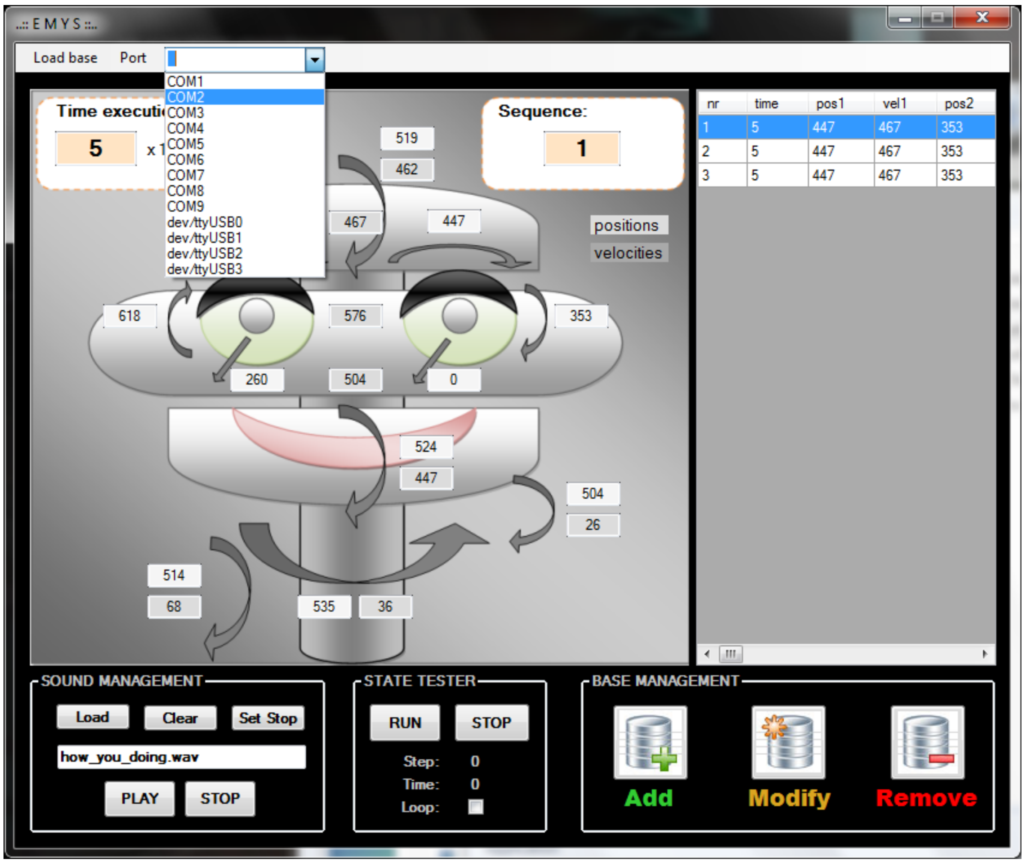
Compiler: Microsoft Visual Studio 2008

Usage:

```
>> Emys.exe
```

Example: The typical work with this application is described by the following steps.

1. Use `Load state` from the top menu to load a `clear` state sheet from `base` folder (Note the `base` folder has to be in the same directory as `Emys.exe`).
2. Choose the port that Emys is connected to via `PORT` combobox. Open the chosen port.
3. Set the desired positions and velocities for each of the head joints. Change the desired position for each joint by clicking light gray box near the joint. The adequate slider will pop up and enable to control joint portioning. To modify the joint velocity click the dark gray box – the control slider will pop up.



nr	time	pos1	vel1	pos2
1	5	447	467	353
2	5	447	467	353
3	5	447	467	353

Figure 3.1: `EMYS` application graphical interface

4. Set the desired duration of the expression in the `Time execution` box located in the upper left corner.

   INFO: If movement duration is too short the robot will skip it to the next expression without reaching the set positions.

5. Input the sequence number in the upper section of the screen. The numbers have to be consecutive integers.

   INFO: If the application reaches a gap in the section numbers (e.g. 3,5) it stops executing the program.

6. It is possible to add sound to choose expressions with `SOUND MANAGEMENT` section. To load a file click `LOAD` button and choose the *.**WAV** file. To stop playing previous sound press `Set Stop` button .

7. To add the current face expression to the list click `Add` in the `BASE MANAGEMENT`

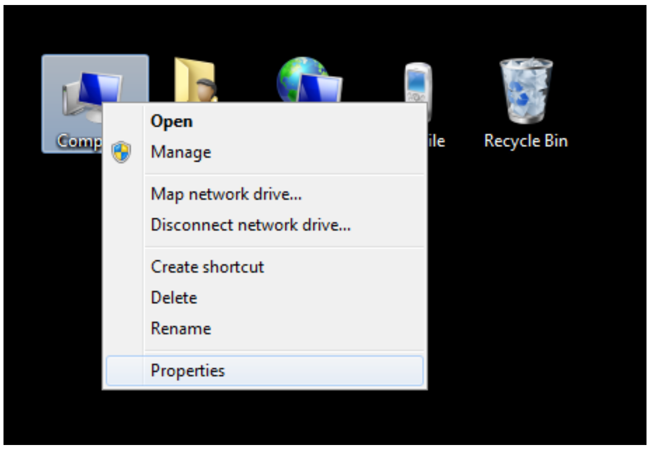
section.

8. Repeat steps 4–7 to add more face expressions.
9. Start face expressions performance by clicking `PLAY` in the `STATE TESTER` section.
10. Individual expressions can be modified or removed via the `BASE MANAGEMENT` section.
    INFO: When an expression is removed from the list, a gap in the section numbers might occur, which will cause the robot to stop executing the program.

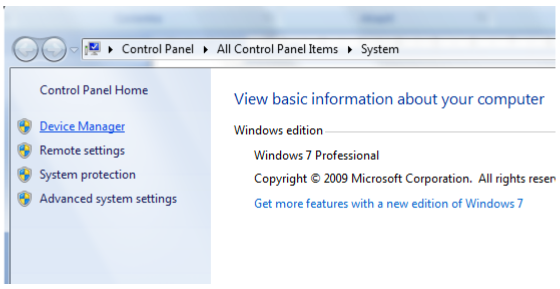
3.2 Changing port name in Windows

If is often observed in the case of Windows, that after connecting a new USB–Serial Converter it is installed as a COM device with very high number. In such a case it is advised to assign a fixed name to such the device. The following instruction describes how to do this.

- Right click `My Computer` and select `Properities`



- Select the `Device Manager` tab.



View basic information about your computer

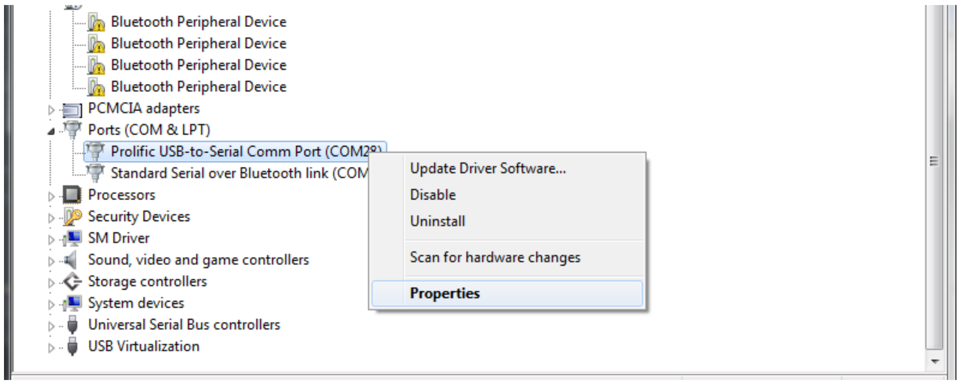
Windows edition

Windows 7 Professional

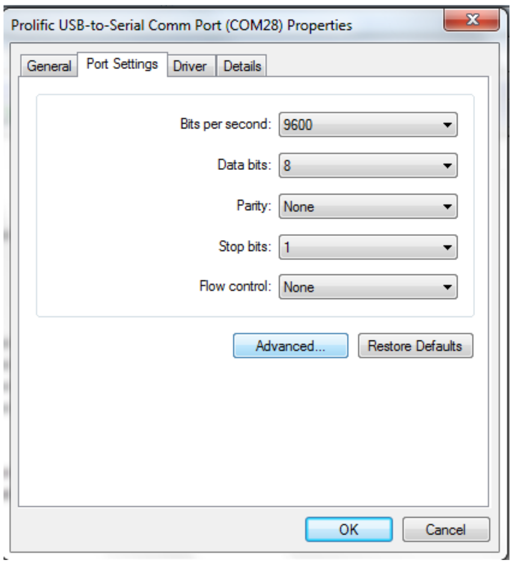
Copyright © 2009 Microsoft Corporation. All rights reser

Get more features with a new edition of Windows 7

- Find the device group `Ports(COM & LPT)`. Select the newly installed device. In this case it will be `Profilic USB-to-Serial Comm Port`. Click the right mouse button over selected device and choose `Properities`.


Bluetooth Peripheral Device
Bluetooth Peripheral Device
Bluetooth Peripheral Device
Bluetooth Peripheral Device
PCMCIA adapters
Ports (COM & LPT)
Prolific USB-to-Serial Comm Port (COM28)
Standard Serial over Bluetooth link (COM
Processors
Security Devices
SM Driver
Sound, video and game controllers
Storage controllers
System devices
Universal Serial Bus controllers
USB Virtualization
Update Driver Software...
Disable
Uninstall
Scan for hardware changes
Properties

- Choose `Port Settings` tab and select `Advanced` button


Prolific USB-to-Serial Comm Port (COM28) Properties

Bits per second: 9600

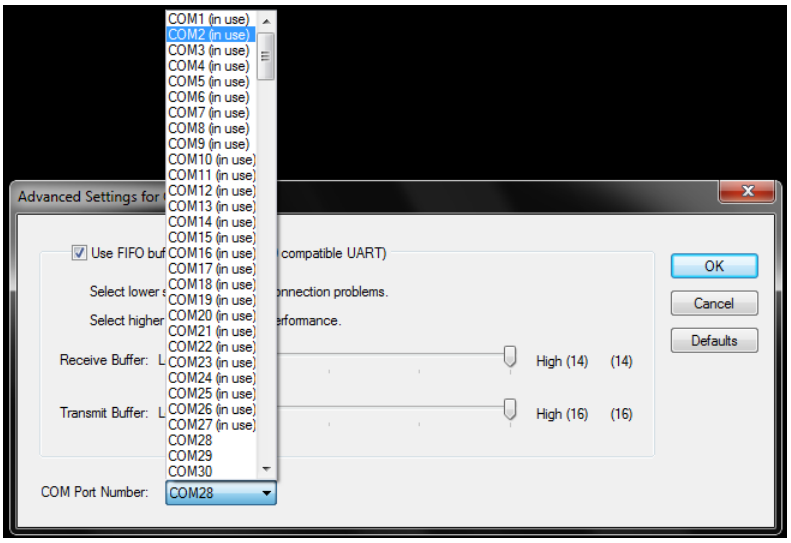
Data bits: 8

Parity: None

Stop bits: 1

Flow control: None

- In this window it is possible to select desired port name for this device.



COM Port Number: COM28

Note: Sometimes system marks ports as `(in use)`. This does not have to mean that this name is reserved. Moreover it does not mean that this device in deed in use. If we are sure that such a device is not connected at the moment to the computer we can select it.

3.3 Linux

3.3.1 **`very_simple_EMYS`, `simple_EMYS`, `EMYS`**

Programs `very_simple_EMYS`, `simple_EMYS`, `EMYS` compiled by the C# compiler in Windows can be run directly in Linux after installing `mono` library. After installing `monodevelop` it is also possible to compile C# examples directly in Linux and using them both in Windows and Linux. Note that it might be essential to install `mono-winforms` library. The only important difference in usage of those applications in Linux is the name of the port. Instead of `COM[number]` they are named `/dev/ttyUSB[number]`.

3.3.2 **`Dynlink` library**

Additionally for Linux system there is provided a C++ library named `Dynlink` and examples of its application. This library allows for communication with Dynamixel servos. It allows both, servos control and configuration. This library is described in detail in separate document.

3.3.2.1 Requirements

The Dynlink library requires the Boost library in version 1.36.0. The current package has not been tested with higher versions of Boost. The home download page of Boost is: http://www.boost.org/

The Dynlink need CMake to build binaries from sources. The home download page of CMake project is: http://www.cmake.org/

To build documentation you need Doxygen which is available in home page of Doxygen project: http://www.stack.nl/ dimitri/doxygen/index.html

Those packages are usually available in all Linux distributions and for Windows platforms.

3.3.2.2 Compilation and installation

The simplest way to build the Dynlink is a sequence of commands in root of project directory:

```
>> mkdir build
>> cd build
>> cmake ../ (see configuration)
>> make
>> make doc (optional)
>> make install (optional)
```

3.3.2.3 Configuration

Before compilation you are able to configure some parameters of this project. To do this use ccmake or cmake-gui tool. The most important parameters are:

CMAKE_INSTALL_PREFIX – location where Dynlink library will be installed

CMAKE_BUILD_TYPE – (available in advanced mode) you can choose between Release or Debug.

After configuration you can start build the project by typing

```
>> make
```

After build the `Dynlink` in build directory you can find subdirectories doc, example, src.

3.3.2.4 Contents

doc/html – this directory contains documentation of Dynlink library in html format.

example – this directory contains two examples which show how to use Dynlink class.

src – this directory contains static library with header which can be used in your projects.

3.3.2.5 Examples

`Dynlink` library is delivered with two examples. First one presents how to open connection with serial port and how to add several actuators to `Dynlink` network. The source code of this example is illustrated in Listing 3.1.

This example starts from defining the serial port parameters like device name `/dev/ttyUSB0` and baudrate `57600`. The program creates main `Dynlink` object name `dynlink` and open communication with it. After establishing communication it tries to add 6 servos with ID number from 0 to 5.

Listing 3.1: Dynlink library connection initialization example

```
/* example1.cpp
 *
 *  Created on: 30-11-2010
 *      Author: aoleksy
 */
/**
 * This example shows how to open connection with serial port
 * and add several actuators to object of Dynlink class.
 */
#include "Dynlink.h" //Main header of Dynlink class
#include <boost/cstdint.hpp> //Defines uint8_t and uint16_t
                             //used in Dynlink class.
#include <string>
#include <iostream>
```

```
#include <cstdlib>

using namespace std;
using namespace dynlink;

string deviceName = "/dev/ttyUSB0"; //Serial device.
int baudrate = 57600; //Baud rate of serial device.
uint8_t fromID = 0x00; //First id
uint8_t toID = 0x05; // Last id

int main(int argc, char* argv[]) {
  Dynlink dynlink; //Main object of Dynlink class.
  if (argc != 1 && argc != 3 && argc != 4 && argc != 5) {
    cerr << "Wrong number of arguments" << endl;
    cout << "Usage:\n\texample1 [fromID toID [pathtodevice [↩
        baudrate]]]" << endl;
    exit(-1);
  }
  if (argc >= 3) {
    fromID = static_cast<uint8_t>(atoi(argv[1])); //get first id
    toID = static_cast<uint8_t>(atoi(argv[2])); //get last id
  }
  if (argc >= 4)
    deviceName = argv[3]; // get device path
  if (argc == 5)
    deviceName = argv[4]; // get baud rate
  if(!dynlink.open(deviceName, baudrate)) { //open serial device
    cerr << "Can't open device " << deviceName << ". " << endl;
    exit(-1);
  }
  Data8 ids(0); //Empty array of actuators id.
  /* Add actuators with id from 0x00 to 0x05. */
  for (uint8_t id = fromID; id <= toID; ++id) {
    cout << "Try to add actuator id=" << static_cast<int>(id) << ↩
        endl;
    if(dynlink.addServo(id)) { // if actuator exists
      cout << "Found actuator at id=" << static_cast<int>(id) << ↩
          endl;
      ids.push_back(id); // Collect found actuators
    }
  }
  cout << ids.size() << " actuator(s) are found." << endl;
  dynlink.close(); // Close connection
  return 0;
}
```

The second example presents how to control a single servo. Similar to the previous example it opens serial port `/dev/ttyUSB0` with baudrate `57600`. Next it adds to `Dynlink` network a servo with ID specified by user. After success in adding the servo program sets a servo speed. After setting new servo position the motor should start acting. Finally program waits until the servo finishes its movement.

Listing 3.2: Dynlink library servo control example

```
/*
 * example2.cpp
 *
 *  Created on: 07-12-2010
 *      Author: aoleksy
 */
/**
 * This example shows how to communicate with the actuators.
 */
#include "Dynlink.h"
#include <boost/cstdint.hpp>
#include <string>
#include <iostream>
#include <cstdlib>

using namespace std;
using namespace dynlink;

/* Path or name of serial device. */
string deviceName = "/dev/ttyUSB0"; //Path or name of serial ↩
    device.
int baudrate = 57600; //Baud rate of serial device.
uint8_t id; // Id of actuator
uint16_t speed; // Speed of movement
uint16_t position; // Goal position

int main(int argc, char* argv[]) {
  Dynlink dynlink; //Main object of Dynlink class.
  if (argc < 4 || argc > 6) { //Control numbers of arguments
    cerr << "Wrong number of arguments." << endl;
    cout << "Usage:\n\texample2 id position speed [pathtodevice [↩
        baudrate]]" << endl;
    exit(-1);
  }
  id = static_cast<uint8_t>(atoi(argv[1])); // Get id number
  position = static_cast<uint16_t>(atoi(argv[2])); // Get position
  speed = static_cast<uint16_t>(atoi(argv[3])); // Get speed
  if (argc > 4)
    deviceName = argv[4]; // Get path to device
  if (argc == 6 )
    baudrate = atoi(argv[5]);
  if (!dynlink.open(deviceName, baudrate)) { // try open serial ↩
      device
    cerr << "Can't open device " << deviceName << ". " << endl;
    exit(-1);
  }
  if (!dynlink.addServo(id)) { // try to add actuator with id
    cout << "Actuator with id " << static_cast<int>(id) << " does ↩
        not exist." << endl;
    dynlink.close();
    exit(-1);
  }
  dynlink.setMovingSpeed(id, speed); // Set speed
  dynlink.setGoalPosition(id, position); // Set position
  cout << "Actuator " << static_cast<int>(id)
      << " is moving to position " << static_cast<int>(position)
      << " with speed " << static_cast<int>(speed) << endl;
  uint8_t moving = 0;
  do {
    dynlink.getMoving(id, &moving);
  } while(moving);
  cout << "Actuator has reached the target position." << endl;
  dynlink.close();
  return 0;
}
```

4

Support

More information about Emys can be found on Emys webpage

http://emys.lirec.ict.pwr.wroc.pl

After login on this page 4.1 a section `Support` will appear. In this section one can find drivers, manual, examples and libraries related with Emys.

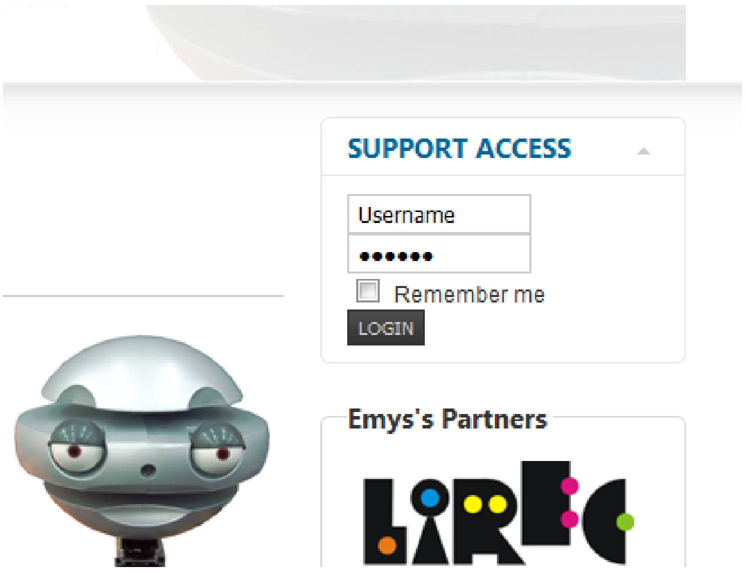


Figure 4.1: Emys webpage login section

Each user has its individual login and password. Your login and password for Emys webpage can be found bellow:

For support please contact with authors of Emys prototype at Wroclaw University of Technology. The contact persons are:

jan.kedzierski@pwr.wroc.pl for hardware related questions.

adam.oleksy@pwr.wroc.pl for software related questions.