ROBOTA

Robot and Simulator

USER GUIDE

Aude Billard, EPFL

In Collaboration with
Jean-Daniel Nicoud, DIDEL
Sylvain Calinon, EPFL
Olivier Michel, Cyberbotics
Andre Guignard, EPFL
Forewords

Robota is the name attached to a family of mini-humanoid doll-shaped robots, created by Aude Billard. The first prototype was presented in August 1998 at the occasion of the 7th SAB conference, held in Zurich. Robota was originally developed as an example of robotic application of artificial intelligence technics [A. Billard. Industrial Robot Journal, 26:1, pp.59-66, Jan 1999]. Robota is currently being used in different research projects that investigate the use of toy robots for children with disabilities ["Games children with autism can play with robota, a humanoid robotics doll". Dautenhahn, K. and A. Billard. 1st Cambridge Workshop on Universal Access and Assistive Technology (CWUAAT), March 2002]. Robota is featured in on-line children's Internet guides [Finditquick, Weigl Educational Publishers, 2000; The SuperSite for Kids, Bonus.com,2001]. Ten Robota robots are being used in an undergraduate robotics class at the University of Southern California (USC). Robota was presented at different science fairs, and is currently being displayed at the French national science museum, La Cite des Sciences et de l'Industrie in Paris [L'homme transforme, November 2001-July 2003]. Scientific publications and media coverage of Robota are available on the Robota Website: www.didel.com/robota.html

The Robota robot is a product of DIDEL SA, www.didel.com. The tutorial and software examples presented in this UserGuide are freeware. Webots-Robota is a simulated version of Robota, developed by Olivier Michel, and based on the Webots simulator. The Webots simulator is a product of Cyberbotics SA, www.cyberbotics.com that offers a simulated world for a number of robots, aside from Robota, such as the Khepera and the Koala robots and the Pioneer robots. Ten Webots licenses are included in the Robota Educational-Pack.

Note for the user

This UserGuide is meant for educators, researchers and hobbyists, interested in programming the Robota robot and its simulator. The UserGuide will get you started using the robot Robota and the Robota-Webots simulator through 6 tutorials. The tutorials follow the assignments of the laboratory sessions of the Robotics undergraduate class, taught by Aude Billard at the University of Southern California in the spring 2002. The tutorials are meant to be used by other robotics class that uses the robot Robota or the Webots-Robota simulator as experimental support. This UserGuide is organized as follows:

Section 1 describes the electronic and mechanical components of the robot Robota.

Section 2 explains the basic user commands to the robot Robota.
Section 3 contains tutorials to program the robot–PC interface (Visual C++ programming environment). The tutorials explain how to control the Robota motors, how to read the Robota sensors, how to develop a speech interface and how to interpret the input from a QuickCam camera to develop a vision-based tracking system.

Section 4 presents an example of programs in C to interface the robot Robota with the Kameleon 376 Single Board Computer. The Kameleon board is a product of K-Team, www.k-team.com, and Must be purchased separately from the Robota Basic Kit and the Robota Educational Kit.

Section 5 contains tutorials to program the Robota-Webots simulator and to interface the simulator with the Robota robot (Visual C++ programming environment). The reader is expected to have minimal knowledge in electronics, in programming in C and C++ and in using the windows Visual C++ programming environment.

Thanks

Aude Billard is very grateful to all the people who contributed to the development of the robot Robota, the simulator Robota, the sample applications of Robota, the Robota User Guide and the Robota website. These people include Auke Jan Ijspeert, Elisabeth Calleja (Research Assistant at the University of Southern California), Mika Satomi (freelance designer), Olivier Carmona (LAMI-EPFL), Rene Beuchat (LAMI-EPFL), Yuri Lopez De Meneses (LAMI-EPFL), Alexander Colquhun (University of Edinburgh), Marie-Pierre Lahalle (La Cite Des Sciences et l'Industrie), Kerstin Dautenhahn (University of Herfordshire), Gillian Hayes (University of Edinburgh), Sharon Demuth (University of Southern California) and many others.
# TABLE OF CONTENT

1 **Robota Components** .................................................................................................................. 5
   1.1 **Unpacking and Inspection** ....................................................................................................... 5
   1.2 **The Robota Robot** .................................................................................................................. 5
   1.3 **Safety Procedures** .................................................................................................................... 6
   1.4 **Robota Motor Board** .............................................................................................................. 7
   1.5 **Robota Sensor Board** ............................................................................................................. 8
   1.6 **Robota External Boards** ......................................................................................................... 9

2 **Getting Started with the Robota-PC interface** ................................................................. 11
   2.1 **PC-robot serial interface** ....................................................................................................... 11
   2.2 **SPI Communication** ............................................................................................................. 14

3 **Robota PC Version - TUTORIALS** ......................................................................................... 16
   3.1 **Visual C++ templates** : ........................................................................................................ 16
   3.2 **Tutorial 1 – Robota Dances and Plays Music** ...................................................................... 20
   3.3 **Tutorial 2 – Robota Speaks** .................................................................................................. 28
   3.4 **Tutorial 3 – Camera : Robota Imitates** ................................................................................. 33

4 **Interfacing Robota with the 376 Kameleon Board:** .......................................................... 40
   4.1 **Tutorial 4: Robota Drives a Car** ......................................................................................... 45
      4.1.1 How to interface Robota’s motors with the car’s motors: ............................................. 46
      4.1.2 A simple neural network to control the avoid obstacle behavior: ....................... 48

5 **Programming The Webots – Robota Simulator** ................................................................. 54
   5.1 **Webots-Robota TCP/IP connectivity** .................................................................................. 54
   5.2 **Tutorial 5: Webots-Robota Vision-Based Imitation game** .............................................. 57
   5.3 **Tutorial 6: Webots-Robota Learning game** ...................................................................... 57

6 **PocketPC – Version of Robota** .......................................................................................... 63
   6.1 **PocketPC : Guideline for Software Installation** ................................................................. 63
      6.1.1 Installation of the programming environment, required for all tutorials: .................. 63
      6.1.2 Installations required for Tutorial 8: .............................................................................. 64
      6.1.3 Installations required Tutorial 9: .................................................................................... 64
      6.1.4 Basic Files of the RobotaPocketPC Workspace: ......................................................... 64
      6.1.5 To Compile and Download Files to PocketPC and Emulator: .................................. 66
   6.2 **Tutorial 7 – Serial Port Communication** .......................................................................... 67

Copyright: A.G. Billard, 2002
LIST OF TABLES AND FIGURES

Table 1: Robota Robot: Technical Specifications .................................................. 6
Table 2: Robota Accessories used in the tutorial examples .................................. 6
Table 3: Motors and Potentiometers - Hexadecimal Commands ............................. 12
Table 4: Summary of Robota commands ................................................................ 13
Table 5: Format of the SPI communication. ............................................................ 14

Figure 1: Robota Motor Board - Front View ........................................................... 7
Figure 2: Robota Sensor Board attaches on the back of Robota Motor Board ....... 8
Figure 3: Robota Sensor Board - Technical Specifications ................................. 9
Figure 4: Robota battery set attaches to the back of the Motor Board ............... 10
Figure 5: The K-Team Kameleon Board attaches to the front of the Motor Board ................................................................. 10
Figure 6: Hyperterminal configuration to communicate with the Robot Robota .... 11
Figure 7: Robota External Keyboard ..................................................................... 40
Figure 8: Robota leg motors are wired to the remote control car motors .......... 47
Figure 9: The Kameleon Board is attached solidly on the front of the Robota motor card.
   The serial cable running from the Kameleon board to the Robota motor board being
   too long is wired around Robota body ................................................................. 48
Figure 10: One layer feedforward neural network to control obstacle avoidance
   behavior ............................................................................................................... 48
Figure 11: Robota sitting in the remote controlled car - Pretty look for the driver of CarBota ................................................................. 49
1 Robota Components

1.1 Unpacking and Inspection

Open the package and identify all the items of your Robota kit.

THE BASIC ROBOTA KIT

The basic Robota kit comprises the following elements:

- 1x Robot Robota: 1x motor board, 1x sensor board, 6x body switches.
- 1x Serial cable.
- 1x Power Supply (120/220V → 12V, 4A)
- 1x Battery set + 1x set of fixation sticks and screws
- 2x screwdrivers
- 2x pair of infra-red emitter/receiver
- 1x Robota UserGuide + 1x Tutorial CD-Rom

THE EDUCATIONAL ROBOTA KIT

The educational Robota kit comprises the following elements:

- 1x Robota basic kit (DIDEL SA)
- 10x Webots licenses (Cyberbotics SA)
- 1x Webots Reference Manual + UserGuide (Cyberbotics SA)

1.2 The Robota Robot

The Robota robot has five motors to drive its two legs, two arms and head. It has 6 switches integrated in its body (in the hands, on the back of the legs, in the neck and on the top of the head). Robota electronic components consist of a Motor Board and a Sensor Board. The Motor board is addressed via a RS232 serial interface from a PC or an external microcontroller. The Sensor Board is addressed via a SPI (serial peripheral interface) serial interface. An external keyboard can also be attached through an extra SPI port on the sensor board, see Section 4. The Motor Board acts as the master of the SPI connection with Sensor Board and external keyboard. Motor Board, Sensor Board and external keyboard are controlled by a PIC microcontroller. A list of technical specifications concerning the robot Robota are given in Table 1. Table 2 lists the accessories used in the tutorial examples of Section 3.

Copyright: A.G. Billard . 2002
Table 1: Robota Robot: Technical Specifications

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Height</strong></td>
<td>45cm</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>14cm</td>
</tr>
<tr>
<td><strong>Weight</strong></td>
<td>~1500gr</td>
</tr>
<tr>
<td><strong>Degrees of Freedom</strong></td>
<td>5 : left /right arm/leg and head</td>
</tr>
<tr>
<td><strong>Actuators</strong></td>
<td>Motorola DC Motors (Maxon A-max 26 mm) with clutches and position potentiometers</td>
</tr>
<tr>
<td><strong>Microprocessors Motor Card</strong></td>
<td>PIC – 16F870, 4MHz</td>
</tr>
<tr>
<td><strong>Microprocessors Sensor Card</strong></td>
<td>PIC – 16F84, 16MHz</td>
</tr>
<tr>
<td><strong>Sensors</strong></td>
<td>Switches (x6), infra-red emitter/receptor (x2), inclinometer (x1), buzzer (x1)</td>
</tr>
<tr>
<td><strong>Battery</strong></td>
<td>7.2V, 7x1.2NiCd</td>
</tr>
<tr>
<td><strong>Running Time:</strong></td>
<td>~30 minutes</td>
</tr>
<tr>
<td><strong>Power supply</strong></td>
<td>12V, 4A.</td>
</tr>
</tbody>
</table>

Table 2: Robota Accessories used in the tutorial examples

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kameleon K376SBC Board (K-TEAM SA)</strong></td>
<td><strong>CPU/OS:</strong> 68376 Motorola / gcc Linux - C with KProject for Win2000</td>
</tr>
<tr>
<td><strong>PC Interface:</strong></td>
<td><strong>OS:</strong> Win 2000, Visual C++ 6.0, Serial Port</td>
</tr>
<tr>
<td><strong>Accessories</strong></td>
<td>QuickCam Camera</td>
</tr>
<tr>
<td></td>
<td>Speech Synthesizer</td>
</tr>
<tr>
<td></td>
<td>Electronic KeyBoard (Xylophone + Joystick)</td>
</tr>
<tr>
<td></td>
<td>Drawing Pad/Touch Screen</td>
</tr>
</tbody>
</table>

1.3 Safety Procedures

Before using any of Robota electronic board, check the operating voltage of the power supply.

If you use a power supply not provided by DIDEL, check that your power supply has a voltage identical with that of this documentation.

When plugging the robot into the power outlet, make sure that the switch lever (on the left handside of the robot) is in the OFF-position (lower position).

Switch off the Robot and the power supply if you do not intend to use the robot for longer than a day.

DIDEL assumes no responsibility for the use of the Robota Robot in any application, including those described in the tutorials of this UserGuide.
1.4 Robota Motor Board

The Robota *motor board* supports Robota five motors and five associated potentiometers. The on-board PIC microcontroller controls the motors’ position and speed see Tutorial 1 in Section 3. Note the parts shown in Figure 1.

![Figure 1: Robota Motor Board - Front View](image)

Robota can be fixed on a stand with two screws, using the two fixation points on the middle of the Motor Board, see Figure 1.
1.5 Robota Sensor Board

The Robota Sensor Board interfaces digital and analogue sensors. Note the parts shown in Figures 2 and 3. The Sensor board provides entries for six body switches (2 switches inside the palms, 2 switches on the back of the ankles, 1 switch in the neck and 1 switch on the top of the head), for 2 analog inputs (e.g. Infra-red receiver) and for 2 analog outputs (e.g. infra-red emitter). The in-built PIC microcontroller controls the buzzer output (eight notes of the scale) and communicates with the motor board via SPI (Serial Peripheral Interface). A second SPI connector is used to connect additional boards.

The switch lever is located on the left handside of the motor board (when the robot faces you), i.e. on the right handside of the motor board when viewed from behind as in Figure 2. To switch on the robot, bring the lever to the upper position.

---

**Figure 2: Robota Sensor Board attaches on the back of Robota Motor Board**
1.6 Robota External Boards

The Robota set of battery has to be mounted on the back of the Motor Board using the fixation sticks provided in the Robota set, see Figure 4. An external microcontroller, such as the K-TEAM Kameleon card or the Compaq Pocket PC IPAQ, can be mounted on the front of the Robota motor card, using the fixation sticks provided in the Robota set, see Figures 4 and 5. The Kameleon board and the Motor board communicate via the USART serial port on the Kameleon board and the 2nd serial port on the back of Robota Motor Board, see Figure 1. Section 4, shows an example of how to interface the K-Team Kameleon board with the Robota Motor Board. Note that the Kameleon Board is not provided by DIDEL SA and must be purchased separately from K-Team at www.k-team.com.
Fixation points on Motor Board

Fixation points on Battery board

Figure 4: Robota battery set attaches to the back of the Motor Board

Fixation points on Kameleon Board

Power supply connector of Kameleon Board. Connect to Robota battery set to

Kameleon Board RS232 connector to PC

USART connector to Robota motor board

Attachment points of the Kameleon Board on Robota Motors board

Figure 5: The K-Team Kameleon Board attaches to the front of the Motor Board
2 Getting Started with the Robota-PC interface

2.1 PC-robot serial interface

Robota serial cable must be connected from one of the PC serial port to the PC serial port of Robota motor board, see Figure 1. Note that Robota motor card has 2 serial entries. The right one (painted in Red) is the PC one, see Figure 1 for the correct positioning. The left one can be used to communicate with an external board, such as the Kameleon board.

It is always a good practice to test that the Robota microcontrollers and the serial connection are working properly before use and while debugging your PC code. You can do this using the Windows Hyperterminal software. To create a Hyperterminal session, go to:

Start ➔ Programs ➔ Accessories ➔ Communication ➔ Hyperterminal

Configure the session « Robota » on COM1, with baudrate 9600, data bits:8, parity :none, stop bites :1, flow control : none.
Robota is switched on by lifting up the lever-switch, located on the right side of the robot. When the robot is being switched on, an **OK** is sent through the RS232 port. The red LED flashes. The green led is on. Allow a 1 second delay robot for the PIC to reset between two consecutive **switching off** and **switching on** procedures.

![Motor numbering (left) and Potentiometers commands (right)](image)

Each robot Robota is provided with 5 motors for the head, left and right arms and left and right legs. To direct a motor to a specific position, you need to send four commands that specify:
- the speed (0..7), maximum speed is 7.
- the motor address (see Table 1)
- the position as a function of the potentiometers (00…90)
- the motor position address (see Table 1).

The hexadecimal commands to the motors and potentiometers are given in table 3.

**Table 3: Motors and Potentiometers - Hexadecimal Commands**

<table>
<thead>
<tr>
<th>@adressing</th>
<th>1 Head</th>
<th>2 Left Arm</th>
<th>3 Right Arm</th>
<th>4 Left Leg</th>
<th>5 Right Leg</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Speed Command</strong></td>
<td>q</td>
<td>a</td>
<td>e</td>
<td>i</td>
<td>m</td>
</tr>
<tr>
<td><strong>Request Position</strong></td>
<td>s</td>
<td>c</td>
<td>g</td>
<td>k</td>
<td>o</td>
</tr>
<tr>
<td><strong>Request Potentiometer</strong></td>
<td>r</td>
<td>b</td>
<td>f</td>
<td>j</td>
<td>n</td>
</tr>
</tbody>
</table>

For instance, if you want to send motor 1 to position 40 on the potentiometer with a speed 2, you will send the command:

**2q40s**

You can verify that the motor has reached the correct position by looking at the reading of the corresponding potentiometer. To request the potentiometer value, you must send the commands given in the 3\textsuperscript{rd} line of Table 3. For instance, to
request the reading of potentiometer 1 (that will give you the position of the motor 1), send the command: “r”. For the reading of potentiometer 2, send the command “b”. The command “u” stops all motors.

To read the state of the six switches located on Robota body, send the commands “[“. The Motor Board returns the state of the switches on two bytes that must be decoded bit by bit, see Tutorial 1 for an example of how to read the switches with a C application.

To play one of the eight notes of the scale with the buzzer, send the command “[“ followed by a number comprised between 8 and 18 (8 for Do/C, 9 for Re/D, 10 for Mi/E, etc.). E.g. to play Do, you should send the command “8[“.

Try first the commands using Hyperterminal to be sure that you understand well how these works, before starting coding! Note that all commands must be typed in lower cases. !!!

Table 4: Summary of Robota commands
2.2 SPI Communication

The communication from Motor Board (master) to Sensor Board and to external Keyboard (slaves) is done through the SPI (Serial Peripheral Interface) protocol. The SPI transfers 3 bytes (transmit mode) and read 3 bytes (receive mode). Table 5 shows the format of the SPI communication.

Table 5: Format of the SPI communication.

<table>
<thead>
<tr>
<th>Transmit Mode</th>
<th>Receive Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor Board xxxx0000 (Tx-IRs + Buzzer)</td>
<td>xxxxxx00 (Body switches)</td>
</tr>
<tr>
<td>Keyboard xxxxxxx (LEDs)</td>
<td>xxxxxxxx (8 Keys)</td>
</tr>
<tr>
<td>Keyboard xxx0000 (Tx-IRs + Buzzer)</td>
<td>xxxxx00 (6 Joystick switches)</td>
</tr>
</tbody>
</table>

The “x” correspond to the relevant bits. The “0” correspond to the irrelevant bits. For instance,

Format in Transmit Mode: 0 0 0 x x x x x

In the Transmit Mode, the last two bits of the five xxxxx commands allow to pulse LEDs at 40kHz. This encoding can be used to pulse two infra-red (IR) emitters (Tx-IR). The encoding is as follows:

- 00000 Tx-IR switched off
- 00001 Tx-IR0 (pins 1-2) active
- 00010 Tx-IR1 (pins 3-4) active
- 00011 Tx-IR0 and Tx-IR1 active

Note that the response time of the TSOP is 0.1ms, leaving enough time for the transfer of two bytes through C routines. A SPI cycle of 8 to 24 bits allows a regular update of the LEDs.

The four last bits encode the status of the buzzer. Melodies are encoded as follows:

- 01000 Beep
- 01001 Bop
- 01010 Do-mi-sol-do
- 01011 Fa-fa-sol-la-re-sol

The corresponding hecadecimal commands 8,9,10-19 generate the 11 notes on the scale and a silence: 8,9,10-19 → do, re, mi … si, do2, re2, mi2, fa2, silence

Each note is played for 0.5sec. In order to repeat a note, a silence must be played between two consecutive commands with the same note.

Format in Receive Mode: sssssyy

In the Receive Mode, the status of the two infra-red receptors (Rx-IR) is encoded on the two last bits of the byte:
The TSOP have not detected a signal
Rx-IR0 is active
Rx-IR1 is active
Buzzer is busy

The SPI sequence to program a distance detector using two pairs of IR Tx/Rx is:

*Activate Tx-IR0 - Read Rx-IR0 – Desactivate Tx-IR0 and Activate Tx-IR1 – Read Rx-IR1 - Desactivate Tx-IR1 and Activate Tx-IR0 – and so forth …*

The status of the body switches (Sensor Board) and the Joystick switches (Keyboard) are encoded on the first 6 bits. The binary encoding is as follows:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>Rx-IR0</td>
</tr>
<tr>
<td>2:</td>
<td>Rx-IR1</td>
</tr>
<tr>
<td>4:</td>
<td>Switch1</td>
</tr>
<tr>
<td>8:</td>
<td>Switch2</td>
</tr>
<tr>
<td>10:</td>
<td>Switch3</td>
</tr>
<tr>
<td>20:</td>
<td>Switch4</td>
</tr>
<tr>
<td>40:</td>
<td>Switch5</td>
</tr>
<tr>
<td>80:</td>
<td>Switch6</td>
</tr>
</tbody>
</table>

The hexadecimal encoding is: 0 1 2 3 4 5 6 7 8 9 : ; < = > ?

Note that the Tx/Rx-Irs, the buzzer and the 6 switches of the Sensor Board and of the Keyboard are addressed with the same encoding.

Tutorial 4 of Section 4 shows an example of C program to pulse and read IR Tx/Rx signals, using the Kameleon board as the microcontroller. The same C routine can be used with the Visual C++ interface presented in Section 3. Tutorial 1 of Section 3 shows an example of C++ routine to read the switches and play notes on the buzzer.
3 Robota PC Version - TUTORIALS

This Section presents 3 tutorials for interfacing Robota with a PC using the Visual C++ programming environment. Tutorial 1 explains how to control Robota’s motors and how to read Robota’s sensors. Tutorial 2 presents an example of how to implement a simple dialogue machine using a speech synthesizer. Tutorial 3 presents an example of how to interface Robota with a camera attached to the PC to implement a video-based imitation game.

3.1 Visual C++ templates:

The Visual C++ project Templates of Tutorials 1, 2 and 3 are available in the Tutorial CD-rom provided with the Robota basic kit. Each template contains the following basic files:
Robota Template Files

Source files
Header files
Resource files
External dependencies

Source files:

Robota.cpp: Definitions of the outlines of the Application.
Robota.rc: Graphical Resources (see explanation below).
RobotaDlg.cpp: Functions for interacting with the Dialog box.
RobotaMotors: Functions to dialogue with Robota motor board and sensor board.
Serial.cpp: Functions for the serial port connection.
StdAfx.cpp: Includes just the standard #includes.

Header files:

Resource.h: Microsoft Developer Studio generated include file.
Robota.h: Main header file for the Robota application.
RobotaDlg.h : Header file for the Dialog box.
RobotaMotors.h: Header file for the RobotaMotors source file.
Serial.h: Header file for the Serial.cpp file
StdAfx.h: File for standard system include files.

Robota Resources
Bitmap
Dialog (Here is the Dialog Box)
String Table
Version

Robota Classes
CRobotaApp
CRobotaDlg
Globals

RobotaMotors.cpp contains general functions for handling strings, functions to initialize the serial port, functions to send motor commands to the Motor Board and functions to send requests to the Motor Board and Sensor Board to read the status of the potentiometers and of the body switches:

GENERAL FUNCTIONS TO HANDLE STRINGS:

char int2str(int i);
Takes an integer as input and returns its character equivalent.
int str2int(char c);
Takes a character as input and returns its integer equivalent.

void concat_string(char string1[],char string2[],char concat[]);
This function concatenates two strings.
int string_length(char string[]);  
Returns the length of a string.

double delta(double x, double y);  
Returns 1 if x>y, 0 otherwise.

FUNCTIONS TO DIALOGUE WITH THE MOTORS AND SENSORS VIA THE SERIAL PORT

void init_serial(void);  
This function initializes the serial port. It specifies the port and baudrate at which to communicate with the board. By default, it sets: hCom=COM1, baudrate=9600.

If you have connected the robot to another port than the COM1 on your PC, change the hCom number in the call
(open_serial(&hCom,"COM1",9600)==0)

void move(int motor_number, int Speed, int Position);  
This function sends a request to the Motor Board to move the motor with the variable motor_number to the position specified by the variable Position with the speed specified by the variable Speed. The PIC controller on the Motor Board proceeds to a Proportional Control, using Speed as the proportional command and Position as the target value. The PIC controller of the Motor Board can be reprogrammed to perform a PID or PD controller. Motor_number is comprised between 1 and 5 (1:Head, 2: Left arm; 3: Right arm; 4: Left leg; 5: Right leg). Speed is comprised between 1 (minimum) and 7 (maximum). Position is comprised between the maximal and minimal values returned by the potentiometer attached to each motor. These vary from one motor to another one. The minimal and maximal values are usually as follows: 0 and 70 for the arms, 0 and 50 for the head and between 0 and 40 for the legs. Refer to Section 2 to learn how to measure these minimal and maximal values using Hyperterminal.

void zero_state(int N);  
This function sends a request to the Motor Board to move the motor with number N to the rest position. The rest position corresponds to the human rest position and not to the zero of the potentiometer. For instance, for the head motor (N=1), the rest position corresponds to the mean value between maximal and minimal value, e.g. 25. You should change the values of the rest positions appearing in the zero_state function, so that these correspond to that of your robot. Refer to Section 2 to learn how to measure the values returned by the potentiometer using Hyperterminal.

int check_potentiometer(int motor_number);
This function sends a request to the Motor Board to read the value of the potentiometer associated with the motor with corresponding motor_number.

```c
void read_switches(int switches[6]);
```

This function sends a request to the Sensor Board (through the SPI port of the Motor Board) to return the status of the 6 body switches on Robota body. The switches are located inside the hands, behind the ankles, on the neck and on the top of the head of the robot. Refer to Section 2.2 to learn how to interpret the hexadecimal values returned by the Motor Board.

`RobotaDlg.cpp` contains functions to manage the dialogue interface on the PC.

```c
BOOL CRobotaDlg::OnInitDialog();
```

This function contains general initialization function. It is called once when the application is launched. You should add your initialization functions here.

```c
void CRobotaDlg::OnDestroy();
```

This function destroys the Dialogue Interface. It is called when the user clicks on the close button on the interface. You should place all commands to kill or close any subsidiary functions or applets here.

```c
void CRobotaDlg::OnTimer(UINT nIDEvent);
```

This function performs a continuous loop with a frequency defined in the global variable LOOP_FREQUENCY.
### 3.2 Tutorial 1 – Robota Dances and Plays Music

This tutorial assumes that you have installed the application winamp. Winamp is a freeware that can play mp3 and wav files. You can download it from [http://www.winamp.com/download/](http://www.winamp.com/download/).

Copy the visual C++ template for this tutorial from the folder `/PC-version/Tutorial1` in the Tutorial CD-Rom of the Robota Kit.

Complete the following steps:

1. Using the serial interface provided by the Hyperterminal software, see Section 2, find out the potentiometer values at the minimal (rest) and maximal positions of each of the five limbs (head, arms and legs). These values vary from one robot to the other. Write down the values corresponding to your robot in your code (20%).
2. Complete the `move` and `zero_state` functions in RobotaMotors.cpp file for each limb using the values found in 1 (20%).
3. Create an `OnDance` routine in RobotaDlg.cpp file, that let the robot dance using its five limbs for 2 minutes (20%).
4. Download a music file (.wav file) from the web. Add a command in the `OnDance` routine to play the music during the dance, using either the winamp software or the playsound windows command. Robota should dance on the rhythmic of the particular music you have chosen (10%).
5. Create a button (Dance) in the user interface that starts the dance routine (5%).
6. Create a “music” button and implement a little melody using the buzzer on Robota sensor card. The command to play a note with the buzzer is a number between 8 and 18 (8 for Do/C, 9 for Re/D, 10 for Mi/E, etc.). Each command is followed by the sign “[”. E.g. to play Do or C, you should send the command “8[”. Create a `OnMusic` function in RobotaDlg.cpp file that sends these commands, using the `send_order_to_robot(hCom,message)` function, defined in RobotaMotors.cpp. (10%).
7. Create a function `response_to_switches` in `Robota_motors.cpp` to let Robota move its arms, legs and head, when the user touches one of the six switches in the hands, legs and head. To read the values of the switches, send first the command “[“ with the `send_order_to_robot(hCom,message)` and, then, use the function `get_answer_from_robot(hCom,answer)` (see the function `check_potentiometer` in file `Robota_motors.cpp`) to read the byte the PIC sends in response. Open hyperterminal and try out the command by pressing different switches to see the hexadecimal response. The byte contains 6 bits with values 1/0 (switch pressed or not). Use the command `bit_answer[i]=answer&0x01;` and `answer=answer>>1;` to read bit by bit. (15%).

**Grading:** 100 points, divided following the above percentages.

---

Copyright: A.G. Billard , 2002
Once you have completed all steps, compare your C++ functions to that given in the folder Tutorial1-solution in the Tutorial CD-Rom of the Robota Kit. Launch the application Robota.exe from the Release directory. The Dialog box for this tutorial should look like this:

![Dialog box for Robota](image)

Pressing the button **Dance** launches a music file and let Robota dance for 1 minute. Pressing the button **Music** starts a little melody played by the Robota buzzer. The **close** button closes the windows application.

**TUTORIAL 1: Solution:**

A: The function `move` in `RobotaMotor.cpp` should be completed as follows:

```c++
switch(motor_number){
    case 1:       // Head
        sprintf(buf,"%iq%is",Speed,Position);
        break;
    case 2:       // Right arm
        sprintf(buf,"%ia%ic",Speed,Position);
```

Copyright: A.G. Billard . 2002
break;
case 3:    // Left arm
    sprintf(buf, "%ie%ig", Speed, Position);
    break;
case 5:    // Left leg
    sprintf(buf, "%im%io", Speed, Position);
    break;
case 4:    // Right leg
    sprintf(buf, "%ii%ik", Speed, Position);
    break;

B: The function zero_state in RobotaMotors.cpp should be completed as follows:

    switch (N){
    case 0: // All limbs at the same time
        message_1 = "4a10c4e10g4i10k4m10o4q25s";
        break;
    case 1: // Head
        message_1 = "4q25s";
        break;
    case 2: // Left Arm
        message_1 = "4a10c";
        break;
    case 3: // Right Arm
        message_1 = "4e10g";
        break;
    case 4: // Left Leg
        message_1 = "4i20k";
        break;
    case 5: // Right Leg
        message_1 = "4m20o";
        break;
    default: // stops all motors
        message_1 = "u";
        break;
    }

C: The function Check_potentiometer in RobotaMotors.cpp should be completed as follows:

    // Send a request to the Motor Board for the potentiometer reading
    if (motor_number == 1) message = "r";
    else if (motor_number == 2) message = "b";
    else if (motor_number == 3) message = "f";
    else if (motor_number == 4) message = "j";
    else if (motor_number == 5) message = "n";
D: The function `response_to_switches` in `RobotaMotors.cpp` should be completed like this:

```cpp
// Read the Sensor Board response
if (get_answer_from_robot(hCom,&answer)==0) {
    // "answer" contains 2 bytes.
    // Read each byte bit by bit.
    for (i=0;i<8;i++){
        bit_answer[i]=answer[0]&0x01;
        answer[0]=answer[0]>>1;
    }
    // The first 4 bits of the first byte contain the state of switches 3, 4, 5 and 6
    switches[2]=bit_answer[0];
    switches[3]=bit_answer[1];
    switches[4]=bit_answer[2];
    switches[5]=bit_answer[3];
    for (i=0;i<8;i++){
        bit_answer[i]=answer[1]&0x01;
        answer[1]=answer[1]>>1;
    }
    // Bits 3 and 4 of the second byte contain the state of switches 1 and 2 (head switches)
    // Values 1 (switch pressed); Value 0 (switches not pressed).
    switches[0]=bit_answer[2];
    switches[1]=bit_answer[3];
    // Move each limb of the corresponding switch is activated
    if (switches[0]==1) move(1,4,25);
    if (switches[1]==1) move(1,4,50);
    if (switches[2]==1) move(2,4,50);
    if (switches[3]==1) move(3,4,50);
    if (switches[4]==1) move(4,4,50);
    if (switches[5]==1) move(5,4,50);
}
else {
    sprintf(buf,"Robota does not answer\n");
    OutputDebugString(buf);
}
```
**DANCE ROUTINE:**

Here is an example of routines that let the robot dance on a music file. The function `OnDance` launches two threads: one plays the music file and the other sends the list of motor commands. Multithreading has the advantage to prevent the applet from freezing while the dance is ongoing. See the complete solution in `RobotaDlg.cpp` and `RobotaDlg.h`.

```cpp
// OnDance: Is activated when button Dance is pressed
void CRobotaDlg::OnDance()
{
    stop_dance = false;
    if (dance_thread_id==NULL)
        dance_thread_id = _beginthread(dance_thread_func,0,(void*)this);
    if (winamp_thread_id==NULL)
        winamp_thread_id = _beginthread(winamp_thread_func,0,(void*)this);
}

// winamp_thread_func: launch winamp software with song.mp3 file in a separate thread
void __cdecl winamp_thread_func(void* param)
{
    BOOL bSoundOn = (BOOL)param;
    if(param) system("winamp  song.mp3");
}

// dance_thread_func: launches a series of motor commands to let the robot dance
void __cdecl dance_thread_func(void* param)
{
    CRobotaDlg* RobotaDlg = (CRobotaDlg*)param;
    Sleep(1200);

    unsigned long  c_ms;
    BOOL arms_state_0 = FALSE;
    BOOL legs_state_0 = FALSE;
    BOOL head_state = TRUE;
    static unsigned long c_correct_0 = 0;
    static unsigned long c_correct_1 = 0;
    static unsigned long c_correct_head = 0;
    static unsigned long c_special_0 = 0;

    int c=clock()/CLOCKS_PER_SEC;
    unsigned long c_start = c;
    bool synch_motion = false;
```
while (!RobotaDlg->stop_dance) {
    
c = clock() / CLOCKS_PER_SEC;
    if ((c - c_start) > (38)) RobotaDlg->stop_dance = true;

    Sleep(5);
    c_ms = clock();

    if ((c_ms > 5000 && c_ms < 15000) ||
        (c_ms > 20000 && c_ms < 25000))
    {
        synch_motion = true;
    }
    else
    {
        synch_motion = false;
    }

    if (c_ms - c_correct_0 > 800)
    {
        c_correct_0 = c_ms;
        if (arms_state_0) // moves left and right arm
        {
            if (synch_motion)
            {
                move(3, 4, 40);
                move(2, 4, 40);
            }
            else
            {
                zero_state(3);
                move(2, 4, 40);
            }
        }
        else
        {
            if (synch_motion)
            {
                zero_state(3);
                zero_state(2);
            }
            else
            {
                zero_state(2);
            }
        }
    }
}
move(3,4,40);

}

}

arms_state_0 = !arms_state_0;


}

if(c_ms-c_correct_1 > 1000) // moves left and right leg
{
    c_correct_1 = c_ms;
    if(legs_state_0)
    {
        if(synch_motion)
        {
            move(4,4,23);
            move(5,4,23);
        }
        else
        {
            zero_state(4);
            move(5,4,23);
        }
    }
    else
    {
        //move(5,4,0);
        if(synch_motion)
        {
            zero_state(4);
            zero_state(5);
        }
        else
        {
            zero_state(5);
            move(4,4,23);
        }
    }
    legs_state_0 = !legs_state_0;
}

if (c_ms-c_correct_head > 1000) // moves head
{
    c_correct_head = c_ms;
    if(head_state)
    {
        move(1,4,00);
    }
    else
```c
{
    zero_state(1);
}

head_state = !head_state;

}
}
zero_state(0);

PLAY A MELODIE:
Here is an example of routines that let the buzzer play a little melodie.

// Play_note: Plays a note and pauses for 200 milliseconds
void play_note(char* n_str)
{
    char note_str[10];
    sprintf(note_str,"%s\n",n_str);
    send_order_to_robot(hCom,note_str);
    Sleep(200);
}

// OnMusic: Play the eight notes of the scale
void CRobotaDlg::OnMusic()
{
    play_note("8");
    play_note("9");
    play_note("10");
    play_note("11");
    play_note("12");
    play_note("13");
    play_note("14");
    play_note("15");
    play_note("0"); // plays a dummy note in order to repeat twice the same
    note, otherwise the PIC ignores the second order
    play_note("15");
    play_note("14");
    play_note("13");
    play_note("12");
    play_note("11");
    play_note("10");
    play_note("9");
    play_note("8");
```
3.3 Tutorial 2 – Robota Speaks

This tutorial requires to have installed the freeware Microsoft speech library SDK 5.1, that can be downloaded from the Microsoft WebSite (www.microsoft.com). SDK 5.1 provides two basic sample voices, Mary and Bob. This software for this tutorial is known to work also with the commercial synthesizers from ViaVoice (www.ibm.com) and ELAN (www.elan.fr).

Copy the visual C++ template for this tutorial from the folder /PC-version/Tutorial2 in the Tutorial CD-Rom of the Robota Kit. The template contains the files Speech.cpp and Speech.h, that were not included in the template of Tutorial 1, see Section 3.1. Speech.cpp contain the functions to launch, interact and terminate the Text_to_Speech (TTS) engine. Speech.h contains header for the file Speech.cpp. The following functions of the file Speech.cpp are relevant to this tutorial:

```cpp
long lPitch, lSpeed;
lPitch defines the pitch of the voice. It takes values between 1 (low pitch) and 350 (high pitch). lSpeed defined the speed of pronunciation. It takes values between 1 (slow) and 300 (fast).

BOOL Text_to_speech( char szSpeak[1024]);
This function passes to the TTS engine the sentence encapsulated in the szSpeak string. Note that the sentence cannot be longer than 1024 character.

BOOL SpeechStart(int choice_language);
This function launches the TTS engine. It takes as parameter choice_language
It should be called only once. If you wish to change voice (for instance, from a female one to a male one) over the course of the application, you need to first call the SpeechTerminate function to kill the TTS thread and then call again the SpeechStart function with the new parameter.

ModeInfo.language.LanguageID
If you use the ELAN TTS Engine, in order to activate the different ELAN voices, this parameter should be changed for one of the following:
LANG_ENGLISH, LANG_SPANISH, LANG_FRENCH, LANG_ITALIAN, LANG_PORTUGESE, LANG_RUSSIAN, LANG_GERMAN

ModeInfo.wGender
If you use the ELAN TTS Engine, this parameter can be changed for one of the following three possibilities:
GENDER_FEMALE, GENDER_MALE, GENDER_NEUTRAL
ModelInfo.wAge = TTSAGE_BABY

This parameter can be changed for one of the following:
TTSAGE_BABY or TTSAGE_ADULT

BOOL SpeechTerminate();

This function kills the TTS engine. It should be called only once the TTS engine has been started with the function SpeechStart.

TUTORIAL 1: EXERCISE

Complete the following steps:

1. Create a user dialogue box IDC_USER_DIALOGUE_BOX and a robot dialogue box IDC_ROBOT_DIALOGUE_BOX.
   To read the user input, use the function
   GetDlgItemText(IDC_EDIT, user_input, sizeof(user_input));
   To write the robot’s answer, use the function
   SetDlgItemText(IDC_ROBOT_DIALOGUE_BOX, str1) 10%.

2. Choose a voice for your robot, by setting different values for the parameters of the ModelInfo structure and the parameters IPitch and ISpeed (in the file speech.cpp) You can test the different voices available by running the software ttsapp.exe (look for the shortcut on the desktop. To test other C++ applets for TTS engine, go to the directory C:/ProgramFiles/Microsoft Speech SDK/TTS/) 10%.

3. Create a crosscheck button to enable or disable the sound 5%.

4. Create a conversational agent: Implement an algorithm that recognizes all words or a subset of the words typed by the user and that let the robot respond appropriately to those words (40%).

5. Let the robot make gestures to emphasize what it says. The robot should engage the conversation by bringing up its favorite topic (Olympic games, travel, politics, etc) 5%.

6. As a demo, implement a predefined ten-sentence exchange with the robot. The robot should also be able to respond to any sentence by picking up relevant words or asking question when it finds no relevant word to prompt an interesting answer 20%.

Grading: 90 points + 10 bonus points for originality.

Once you have completed all steps, compare your C++ functions to that given in the folder Tutorial2-solution in the Tutorial CD-Rom of the Robota Kit. Launch the
application *Robota.exe* from the *Release* directory. The Dialog box for this tutorial should look like this:

![Dialog box](image)

**TUTORIAL 1: Solution**

A: Create a crosscheck button to enable or disable the sound

```c
void CRobotaDlg::OnSound(){
    m_Sound=!m_Sound;
}
```

A: Create a simple conversational agent that can answer to 20 keywords.

```c
// Defines global variables for conversation
#define MAX_CHAR_INPUT 250  // max length for IDC_*_DLG_BOX
conv cvs[20]; int last_cvs; // Pointer to predefined sentences
```
typedef struct conversation {
    char trigger[30];
    char response[100];
    char message[20];
} conv;

void make_trigger(int index, const char *trigger, const char *response, const char *message=0) {
    strcpy(cvs[index].trigger, trigger);
    strcpy(cvs[index].response, response);
    if (message)
        strcpy(cvs[index].message, message);
    else
        strcpy(cvs[index].message, "b");
}

void init_sentences(){
    make_trigger(0, "hello", "Hi! How are you today?");
    make_trigger(1, "play", "Oooh! I love to play! Will you play with me?");
    make_trigger(2, "old", "I'm only 2 years old.");
    make_trigger(3, "understand", "Actually, I lie. I can't understand ANY questions!");
    make_trigger(4, "patty", "Pattycake is my favorite game!", "4e60g4a70c");
    make_trigger(5, "good girl", "No, I'm a naughty girl.", "7q45s");
    make_trigger(6, "food", "I eat only electricity!");
    make_trigger(7, "goodbye", "It was fun talking with you!");
    make_trigger(8, "leg", "Do you think I have pretty legs?", "4m25o");
    make_trigger(9, "good", "Thanks so much. Please, keep talking to me.", "");
    make_trigger(10, "girl", "I am a very pretty girl, don't you agree?", "7q45s");
    make_trigger(11, "drink", "I am too young to have a drink!", "4m25o");
    make_trigger(12, "hair", "My hair are a mess. Can you do something?", "4q45s");
    make_trigger(13, "speak", "I am still learning to speak, but I am progressing!", "4m25o");
    make_trigger(14, "robot", "I am a robot and I would like to meet other robots!", "4m25o");
    make_trigger(15, "human", "I am not a human. Are you a human?", "4m25o");
    make_trigger(16, "no", "I like positive answers.", "7q45s");
    make_trigger(17, "yes", "Yes too!", "7q25s");
    make_trigger(18, "baby", "Baby robots are the best, don't you think?", "4m25o");
    make_trigger(19, "?", "I didn't understand your question. Can you be more specific?");
    make_trigger(20, ", " , "Huh? I'm just a little kid. I don't know too many sentences yet.");
    last_cvs = 20;
}
OnSpeak: Is activated when the user clicks on the OnSpeak button
// Let the robot answer a predefined sentence depending on the keywords found
// in the user query

void CRobotaDlg::OnSpeak()
{
    char user_input[MAX_CHAR_INPUT];
    GetDlgItemText(IDC_USERIALOGUE_BOX, user_input, MAX_CHAR_INPUT);

    for (int i=0; i<=last_cvs; i++) {
        if (strstr(user_input, cvs[i].trigger)) {
            SetDlgItemText(IDC_USERIALOGUE_BOX, cvs[i].response);
            if (strlen(cvs[i].message) > 1)
                send_order_to_robot(hCom, cvs[i].message);
            if (m_Sound) {
                Text_to_speech(cvs[i].response);
            }
            break;
        }
    }
}
3.4 Tutorial 3 – Camera: Robota Imitates

This tutorial requires a Camera. The software for this tutorial is known to work with either a Logitech QuickCam Camera connected through the USB port of the PC, or with a X11 camera connected to a WinTVGo framegraber.

Copy the visual C++ template for this tutorial from the folder PC-version/Tutorial3 in the Tutorial CD-Rom of the Robota Kit. The dialog box looks like this:

The template contains the files wintv.cpp and wintv.h, that were not included in the basic template of Tutorial 1, see Section 3.1. Wintv.cpp contain functions to display and read the camera input. Wintv.h contains header for the file wintv.cpp. The following functions of the file wintv.cpp are relevant to this tutorial:

```cpp
m_pWinTV = new CCapture;
```

m_pWinTV is a handle to the Capture window of the camera. It is initialized when the application is launched by
CRobotaDlg::CRobotaDlg(CWnd* pParent /*=NULL*/) and destroyed when the application is being closed by CRobotaDlg::~CRobotaDlg() in the file RobotaDlg.cpp.

HWND m_hwndParent;
Handle to the parent window, i.e. the main RobotaDialogue window.

bool isRed(int r, int g, int b);
This function returns 1 if r, the number of red pixels, is greater than b, the number of blue pixel, and g, the number of green pixels.

bool isGreen(int r, int g, int b);
This function returns 1 if g, the number of green pixels, is greater than b, the number of blue pixel, and r, the number of red pixels.

int top=46, bottom=274;
Specifie the position of the camera window in the RobotaDialogue Box.

int w_width, w_height;
Specifie the dimension of the camera window in the RobotaDialogue Box.

BYTE* hpPixel = (BYTE*) lpVHdr->lpData;
Pointer to the Video Data. The pointer length is 3*w_width*w_height, as there are 3 (Red Green Blue, RGB) pixels attached to each camera light detector.

int dwWidth = m_nWidth, dwHeight= m_nHeight;
Width and height of the complete pixel image.

void CCapture::DestroyCaptureWindow()
This function destroys the capture window and is called by CRobotaDlg::~CRobotaDlg() i.e. when the dialogue box is closed.

void CCapture::ColorTracking(HWND hwndParent, LPVIDEOHDR lpVHdr);
This function tracks red and green colored objects. It stops Robota motors when it detects a red object (stop flag) and starts Robota motors when it detects a green object.

void CCapture::MotionTracking(HWND hwndParent, LPVIDEOHDR lpVHdr);
This function tracks change of brightness across the whole image.

inline LRESULT CCapture::FrameCallbackProc(HWND hWnd,LPVIDEOHDR lpVHdr);

Copyright: A.G. Billard . 2002
Add to this Callback function the tracking function that you want to run continuously. For instance, comment out ColorTracking or MotionTracking when you want to use only one of the two.

```cpp
BOOL CCapture::init_wintv(HWND hwndParent);
```
Initialize the parameters of the capture window. It is called by the function void CRobotaDlg::OnConnect() in the file RobotaDlg.cpp, i.e. when the user presses the button connect.

```cpp
void CCapture::delete_wintv();
```
This function deletes the capture window. It is called by the function CRobotaDlg::OnDisconnect() in the file RobotaDlg.cpp, that is activated when the user clicks on the button disconnect.

```cpp
void CRobotaDlg::OnTracking()
```
This function starts launches the callback function FrameCallbackProc and is called by the function CRobotaDlg::OnTracking() when the user presses the button Tracking.

**Tutorial 3: Exercise**

Complete the following steps:

1. Complete the Motion Tracking function in wintv.cpp file to track the motions of the left and right arms and of the head of the user, by measuring the variation of brightness in the left and right hand-sides, and the upper part of the image. The camera pixels are stored in an array called hPixel. Each camera pixel correspond to 3 RGB hPixel values (RGB stands for red-green-blue). Follow the example of ColorTracking function in wintv.cpp to find out how to read and interpret the values of the camera pixels (60%).

2. Let the robot move in mirror fashion to the user: i.e. the robot’s left arm moves when the user’s right arm moves. The robot’s head shakes in response to the user head’s shake (20%).

3. Segment each of the 2 tracking areas (for left and right arms) into 3 sub-areas, so that the robot can mimic more closely the motions (i.e. arm up, arm in the middle, arm down) (15%).

4. Take a nice picture of your robot using Logitech QuickCam program and replace the one in the template (5%).
Grading: 100 points.

Once you have completed all steps, compare your C++ functions to that given in the folder Tutorial3-solution in the Tutorial CD-Rom of the Robota Kit. Launch the application Robota.exe from the Release directory. The Dialog box for this tutorial should look like this:

![Image of Robota tutorial dialog box]

**Tutorial 3: Solution**

Here is a solution for the MotionTracking function. See the files wintv.cpp and wintv.h in Tutorial3-solution folder for the initialization of the variables called by this function.

```cpp
void CCapture::MotionTracking(HWND hwndParent, LPVIDEOHDR lpVHdr) {
    BYTE* hpPixel = (BYTE*) lpVHdr->lpData;   // Pointer to Video Data
    int  xpos, ypos;
    int  dwWidth = m_nWidth, dwHeight= m_nHeight;   // Width and height of the image
```
int pixelSizeInBytes = lpVHdr->dwBytesUsed / (dwWidth*dwHeight);  // Size of each pixel (3 bytes, i.e. 24 bits)
int cur_brightness = 0;
int dwHalfWidth = dwWidth/2, dwHalfHeight = dwHeight/2;
int arm_col_width = dwWidth / 4;
int head_row_height = dwHeight / 2;
int head_total_width = dwWidth / 2;

// Define tracking areas; 3x3 areas; 3 left; 3 right and 3 top
area[ARM_LEFT_BOTTOM].left = 0;
area[ARM_LEFT_BOTTOM].bottom = 0;
area[ARM_LEFT_BOTTOM].top = dwHeight / 3;
area[ARM_LEFT_BOTTOM].right = dwWidth / 4;
area[ARM_LEFT_BOTTOM].sum = 0;

area[ARM_LEFT_MIDDLE].left = 0;
area[ARM_LEFT_MIDDLE].bottom = dwHeight / 3;
area[ARM_LEFT_MIDDLE].top = 2 * dwHeight / 3;
area[ARM_LEFT_MIDDLE].right = dwWidth / 4;
area[ARM_LEFT_MIDDLE].sum = 0;

area[ARM_LEFT_TOP].left = 0;
area[ARM_LEFT_TOP].bottom = 2 * dwHeight / 3;
area[ARM_LEFT_TOP].top = dwHeight;
area[ARM_LEFT_TOP].right = dwWidth / 4;
area[ARM_LEFT_TOP].sum = 0;

area[ARM_RIGHT_BOTTOM].left = 3 * dwWidth / 4;
area[ARM_RIGHT_BOTTOM].bottom = 0;
area[ARM_RIGHT_BOTTOM].top = dwHeight / 3;
area[ARM_RIGHT_BOTTOM].right = dwWidth;
area[ARM_RIGHT_BOTTOM].sum = 0;

area[ARM_RIGHT_MIDDLE].left = 3 * dwWidth / 4;
area[ARM_RIGHT_MIDDLE].bottom = dwHeight / 3;
area[ARM_RIGHT_MIDDLE].top = 2 * dwHeight / 3;
area[ARM_RIGHT_MIDDLE].right = dwWidth;
area[ARM_RIGHT_MIDDLE].sum = 0;

area[ARM_RIGHT_TOP].left = 3 * dwWidth / 4;
area[ARM_RIGHT_TOP].bottom = 2 * dwHeight / 3;
area[ARM_RIGHT_TOP].top = dwHeight;
area[ARM_RIGHT_TOP].right = dwWidth;
area[ARM_RIGHT_TOP].sum = 0;

area[HEAD_LEFT].left = dwWidth / 4;

Copyright: A.G. Billard, 2002
area[HEAD_LEFT].bottom = dwHeight / 2;
area[HEAD_LEFT].top = dwHeight;
area[HEAD_LEFT].right = dwWidth / 4 + dwWidth / 2 / 3;
area[HEAD_LEFT].sum=0;

area[HEAD_MIDDLE].left = area[HEAD_LEFT].right;
area[HEAD_MIDDLE].bottom = dwHeight / 2;
area[HEAD_MIDDLE].top = dwHeight;
area[HEAD_MIDDLE].right = dwWidth / 4 + 2 * dwWidth / 2 / 3;
area[HEAD_MIDDLE].sum=0;

area[HEAD_RIGHT].left = area[HEAD_MIDDLE].right;
area[HEAD_RIGHT].bottom = dwHeight / 2;
area[HEAD_RIGHT].top = dwHeight;
area[HEAD_RIGHT].right = 3 * dwWidth / 4;
area[HEAD_RIGHT].sum=0;

ourRect *curRect;

// Defines motor commands associated with each tracking area
// Invert left-right to simulate mirror image when placing the camera
// in front of Robota
strcpy(area[ARM_RIGHT_BOTTOM].message, "4e03g");
strcpy(area[ARM_RIGHT_MIDDLE].message, "4e29g");
strcpy(area[ARM_RIGHT_TOP].message, "4e50g");
strcpy(area[ARM_LEFT_BOTTOM].message, "4a00c");
strcpy(area[ARM_LEFT_MIDDLE].message, "4a40c");
strcpy(area[ARM_LEFT_TOP].message, "4a70c");
strcpy(area[HEAD_LEFT].message, "4q5s");
strcpy(area[HEAD_MIDDLE].message, "4q25s");
strcpy(area[HEAD_RIGHT].message, "4q52s");

// Iterate along the height of the image
for (ypos=0; ypos<dwHeight;ypos++) {

    // Iterate along the width of the image
    for (xpos=0; xpos<dwWidth;xpos++) {

        curRect = 0;
        for (int a=0; a<9; a++) {
            if (isInsideRect(area[a],xpos,ypos))
                break;
        }

        if (a != 9) {     // set green and blue pixel values to zero

Copyright: A.G. Billard . 2002
hpPixel[0] = 0;
hpPixel[1] = 0;
area[a].sum += hpPixel[2];
}
// Update the above if-case to create different if-cases that
// determine each of the required areas.

hpPixel+=3;
}

int left_arm_index = leastAverage(ARM_LEFT_BOTTOM, ARM_LEFT_MIDDLE, ARM_LEFT_TOP);
int right_arm_index = leastAverage(ARM_RIGHT_BOTTOM, ARM_RIGHT_MIDDLE, ARM_RIGHT_TOP);
int head_index = leastAverage(HEAD_RIGHT, HEAD_MIDDLE, HEAD_LEFT);

if (left_arm_last != left_arm_index) {
    send_order_to_robot(hCom, area[left_arm_index].message);
    left_arm_last = left_arm_index;
}
if (right_arm_last != right_arm_index) {
    send_order_to_robot(hCom, area[right_arm_index].message);
    right_arm_last = right_arm_index;
}
if (head_last != head_index) {
    send_order_to_robot(hCom, area[head_index].message);
    head_last = head_index;
}
4 Interfacing Robota with the 376 Kameleon Board:

This tutorial assumes that you have purchased the Kameleon 276 computer board from K-Team (www.k-team.com) and are familiar with the KTProject compiling environment. Follow the instructions of Section 1 to attach the Kameleon Board on the Robota Motor Board. This tutorial includes routines to communicate with the Robota Keyboard via the second SPI connector of the Robota Sensor Board. The Robota Keyboard can be purchased from DIDEL (www.didel.com) and is not part of the Robota Basic Kit. Note that you can use this template without having the keyboard. In this case, simply ignore the two routines (void keys() and void joystick()) in behavior.c that communicates with the keyboard.

Figure 7: Robota External Keyboard.

Copy the PC-version/Template_Kameleon. The Kameleon board is programmed using the KTProject environment. Launch KTProject by clicking on the desktop icon. Choose the mode Kameleon and Open Existing Project. Choose the path to the local copy of the template_kameleon you have created.
Open the project editor, if it is not there by default (file --> Project --> Project editor). You should see the following list of files appear.

![Project Explorer](image)

**Kameleon Template Files**

Update only the following *Source files*:

a) Robota.c: Launch the different processes, start the communication with Robota motor card along the second serial line.

b) Robota.h: header file for Robota.c

c) Behavior.c: Template of functions for interacting with the robot’s sensors.

d) Behavior.h:

To compile, go to **tools --> build.robota.c** This creates a binary file called robota.s37.

To download the file to the Kameleon Processor, do the following: 1) connect the serial port to the Kamelon board, 2) create a hyperterminal session, see below, 3) download robota.s37 as a text file (in hyperterminal, choose **Transfer --> Send Text File**) and then select **robota.s37**. 
Configure the hyperterminal session «Kameleon» on COM2, with baudrate 38400, data bits:8, parity: none, stop bits:2, flow control: none.

Reset the Kameleon board, by pressing on the white reset button (see K-Team instructions). In the hyperterminal window, you should see a reply from the Kameleon board such as «ROM of K376sBC …». Type the command «run sloader» in the hyperterminal window. Then, Download the robota.s37 file by opening the menu «Transfer->Send Text File -> (Select the file robota.s37) ».
The files `Robota.c` and `Behavior.c` contain functions to interface the Kameleon board with the Motor Board of Robota. Here are definitions of these functions.

```c
void tim_suspend_task (int milliseconds);
This function creates a delay defined by the variable passed in its parameters.

void var_change_led (int leds);
This function sets the active status of the leds on the Kameleon board.

void init_tpu();    // Initialize the TPU

void spi_send2(&spi_status[0], 3, beep, leds, music);
This function communicates through the SPI with the Sensor Boards and the external Keyboard (see Section 2.2 for detailed explanations of the SPI communication mode). The parameter `beep` sets the status of the buzzer on the Robota Sensor Card. The parameter `leds` sets the status of the leds on the Keyboard. The parameter `music` sets the status of the buzzer on the Keyboard.
```
void FlushUSART();
This function flushes the USART serial port. The USART serial port must
be flushed before starting to any communication with the Motor Board to
ensure clean data transfer.

void SendUSART(string *message);
This function sends a byte “message” through the USART.

void com_reset();/* Reset communication channel; call at
initialization only */
void var_reset();/* Reset communication variables; call at
initialization only */
void usart_reset(); /* Reset USART channel; call at initialization only */
void usart_config(4); /* Config USART in mode 9600 bps */

In behavior.c:

void keyboard(int keys);
This function plays a note on the Buzzer of the Robota Keyboard for each
key pressed on the Robota Keyboard and light up the LED on top of the
key pressed on the Robota Keyboard
void keyboard(int keys)

void body_interact(int switches);
This function reads the status of the six switches located on Robota body.

void joystick_interact(int joystick);
This function sends and receives infra-red pulses from pairs of infra-red Tx/Rx sensors
connected to the Robota Sensor board.

void joystick_interact(int joystick);
This function reads the status of the 6 Joystick switches located on the
external keyboard.
4.1 Tutorial 4: Robota Drives a Car

Robota and the Kameleon Board have been used in the CarBota project to let Robota control a remote control car autonomously, see photos below.

**Goal:** To create the most high-tech remote control car ever!

**Tools:**
- Remote control car
- Kameleon board
- Robota battery set
- 4 switches
- 2 pairs of IR Tx/Rx

If this tutorial is used in a classroom, it should be accomplished by a team of 2 or 3 students.

**Task:**

1. Connect Robota leg motors to the two motors of a remote control car. The two car’s motors control the forward/backward motion and the left/right turn motion of the car. (30%)

2. Create routines `move_forward`, `turn left`, `turn right`, `move backwards` that send appropriate motor commands to Robota leg motors to let Robota drive the car in each of the directions of motion. (10%)

3. Create a front bumper and a back bumper, by gluing two switches on the front and on the back of the car. Connect the 4 switches to Robota sensor board. Place the two pairs of Infra-red emitter/receptor on the front of the car to provide a distance sensor. Connect the infra-red sensors to Robota Sensor Board, see Section 1 for the emplacement of the connectors. Engineer the car to allow Robota to sit on it in a stable position. Put two switches on the top of the car to provide an on/off switch for the car. (20%)

4. Create a routine `detect_bump` in the file `behaviour.c` that let Robota detect when there is a bump (i.e. when the state of one switch becomes 1) or when there is an obstacle at proximity (i.e. when the IR Rx measure 1 instead of 0). (10%)
5 Create a routine react_to_obstacles that let Robota avoid obstacles by driving the car appropriately in response to a bump or to an obstacle in close proximity. (20%)

6 Let Robota move her arms upwards in response to a bump. Let Robota turn the head in the direction in which the car turns. (10%)

Grading: 100%

During the spring 2002, a team of 3 USC undergraduate students created CarBota. Here is an excerpt of the instructions they were given alongside the project.

4.1.1 How to interface Robota’s motors with the car’s motors:

On the car:
Dismantle and locate the controller board of the remote control car. Using a voltmeter, determine the ground on the car’s circuit. Determine the four Vcc_out pins of the car’s microcontroller, i.e. the pins that send the pwm control signal to each of the two motors. Follow the traces originating from the chip’s four pins to the motors. Once you are certain that you have correctly identified which tracks to apply Vcc_out to power the car’s motors, cut the traces. Solder a red wire from the ground and 4 yellow wires from the 4 ends of the tracks. To test that the 4 functionalities (forward, backward, left and right) of the car remain functional, apply a voltage (using a battery) between the ground (red wire) and each of four Vcc_out points, one at a time!

On Robota:
On each of Robota two leg motors, unsolder the ends of the red and black wires that attach on the motor. If you are to leave the wires out in the air for some time with the robot powered, solder the black and red wire with a 200 Ohm resistor to prevent a shortcut. Then, unsolder the wire, when ready to proceed to the wiring of Robota to the car.
The voltage output to Robota’s motor is 9V. Compare this to the voltage required by the car, and determine the current required by each. The car will be heavier once Robota is sitting on it. If required, create a little circuit to amplify the voltage applied by Robota motors. Solder a long black wire from the ground on Robota motor board that you can take from the power plug on the back of Robota motor board (see Figure 1).

Wire up Robota to the car:
Connect the black wire from the ground of the car circuit to the black wire from the ground of the Robota motor card. Connect the black and red wires of each of the two Robota motors to the corresponding pairs of Vcc/Ground (yellow wires) of the car motors.
Figure 8: Robota leg motors are wired to the remote control car motors

To test if the wiring is correct, open the Hyperterminal session created in Section 2 and send a motor command to each of Robota leg motors. You should see the two motors of the car being driven in both directions. Test different speed command (the speed command determines the torque sent to the motor), and determine which speed command sends a torque sufficient, so that the car manages to move with the added weight created by Robota sitting in the car.

Interfacing the Kameleon Board and the Robota Motor Card:

This section assumes that the reader has read the K-Team leaflet for the 376 Kameleon and is familiar with the instructions required to compile and download programs to the Kameleon board.

Attach the Kameleon board to the front of the Robota Motor Card. Connect the USART serial port of the Kameleon board to the PC serial port of the Motor Card (see Figure 2) through a short RS232 (9 pins) serial cable. By default, the Kameleon board communicates with the PC at a baudrate of 38400. The baudrate must be changed to the 9600 baudrate used by the Robota Motor Board. **Before downloading** the program from the PC to the Kameleon board, type the command “serial 9600” on the prompt of KTerminal (or any other terminal used to communicate with the Kameleon board), then change the terminal setting to 9600 and continue with the usual commands “run sloader”, etc.
Figure 9: The Kameleon Board is attached solidly on the front of the Robota motor card. The serial cable running from the Kameleon board to the Robota motor board being too long is wired around Robota body.

4.1.2 A simple neural network to control the avoid obstacle behavior:

\[ y = \theta \left( \sum_i w_i \cdot x_i \right) \]

Figure 10: One layer feedforward neural network to control obstacle avoidanc behavior

Figure 10 shows the layout and weight values of a one-layer feedforward neural network controller for obstacle avoidance. The network takes as input the state of the left and right bumper switches. The output of the network determines the direction of motion of the motor. Each neuron is modeled as a perceptron:

PERCEPTRON(1940, McCulloch & Pitts)

Threshold Function:

\[ f(z) - \theta(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} \]
The network output is mapped as follows: (1 1): forward; (1 0): Left turn; (0 1): right turn; (0 0): stop

The same network can be expanded to take into account the infra-red receiver inputs and the back bumper switches, by duplicating the connection to the motors. The infra-red-motor connections take the same weights as those for the frontal bumpers. The back bumpers-motor connections take left/right inverse values for the weights. The code for the neural network implementation is given in the file Carbota.c in the Template_Kameleon directory from the Template CD-Rom from the Robota Kit.

**Exercise:** implement a hebbian learning function to let the robot learn the values of the weight.

![Figure 11: Robota sitting in the remote controlled car - Pretty look for the driver of CarBota](image)

```c
#include <stdio.h>

#define Ninputs 2
#define Noutputs 2

int input[Ninputs], output[Noutputs];
double weight[Ninputs][Noutputs];

int running = 0;
int theta(double x, double y);

void init_network();
void update_network();
void display_network();
void activate_network();

void turn_left();
```

Copyright: A.G. Billard, 2002
void turn_right();

void init_network()
{
    // initializing the inputs to zero values
    input[0] = 0;     //LB
    input[1] = 0;     //RB

    //initializing outputs to zero values
    output[0] = 0;    //LT
    output[1] = 0;    //RT

    //initialize weights
    weight[0][0] = -0.9; //from LB to LT
    weight[0][1] = 1;   //from LB to RT
    weight[1][0] = 1;   //from RB to LT
    weight[1][1] = -0.9; //from RB to RT
}

void update_network()
{
    //when we notice that one of the switches gets activated...
    //we need to update the values in the network...
    //...but we also need to update the weights....(to do real learning)

    input[0] = array_switch[0];
    input[1] = array_switch[1];

    //then activate the network to get the outputs
    activate_network();

    //then act on those new outputs from the network

    if(output[0] == 1 && output[1] == 1 && running)
    {
        //TURN AROUND
    }
    else if(output[0] == 1 && running)
    {
        turn_left();
    }
    else if(output[1] == 1 && running)
    {
        turn_right();
    }
}
void activate_network()
{
    float temp;

    //output[0]= (input[0]*weight[0][0]) + (input[1]*weight[1][0]);
    temp = (input[0]*weight[0][0]) + (input[1]*weight[1][0]);

    if (temp >= 0)
        output[0] = 1;
    else
        output[0] = 0;

    //output[1] = (input[0]*weight[0][1]) + (input[1]*weight[1][1]);
    temp = (input[0]*weight[0][1]) + (input[1]*weight[1][1]);

    if (temp >= 0)
        output[1] = 1;
    else
        output[1] = 0;
}

void turn_left()
{
    printf("u\r\n");
    //have CarBota react
    flail();
    //turn wheel to left
    printf("7i40k\n\n");
    //back up
    printf("7m40o\n\n");
    //wait
    tim_suspend_task(1000);
    //stop
    printf("u\r\n");
    //wait
    tim_suspend_task(1000);
    //go forward
    printf("7m00o\n\n");
}

void turn_right()
{  
  printf("u\n\n");  
  //have CarBota react  
  flail();  
  //turning wheel to right  
  printf("7i00k\n\n");  
  //back up  
  printf("7m40o\n\n");  
  //wait  
  tim_suspend_task(1000);  
  //stop  
  printf("u\n\n");  
  //wait  
  tim_suspend_task(1000);  
  //go forward  
  printf("7m00o\n\n");  
}

///////////////////////////////////////////////////////////////////////////
//This function has CarBota react to the bump switches
///////////////////////////////////////////////////////////////////////////
void bumper()
{
    for(EVER)
    {
        if(array_switch[3] == 0)
        {
            running = 1;
            //Send Car Forward
            printf("7m00o");
        }
        if(array_switch[2] == 0)
        {
            running = 0;
            //stop car
            printf("u");
        }
        update_network();
    }
}

//This function has CarBota move her arms and head for use when she bumps
something
void flail(void)
{
        printf("7e58g\n"); //arm
        printf("7a35c\n"); //arm
        tim_suspend_task(500);
        printf("7e25g\n"); //arm
        printf("7a20c\n"); //arm
        tim_suspend_task(500);
}
5 Programming The Webots – Robota Simulator

This section requires that you have purchased and installed the webots simulator and have downloaded the Robota world files from Cyberbotics (www.cyberbotics.com). A licence of webots is included in the Robota Educational Pack, but not in the Robota Basic Kit.

5.1 Webots-Robota TCP/IP connectivity

Copy the two templates for this tutorial from the folder PC-version/Robota-webot/. This folder contains two subdirectories: Robota/ and Robota-client/ Place the two subdirectories Robota/ and Robota-client/ in your Webots/Controllers/ directory.

> Go to the Robota directory. Launch the Robota.dsw project. Compile the project (key F7).

> Launch the application webots from the desktop icon and starts the Robota world : file->open- robota.wbt. The following two applets will appear:
> Go to the Robota-client directory. Launch the Robota.dsw project. Compile and execute the project (key F5). The following user interface will appear:

> The webots application will connect automatically and display the following.
>>> If the webots application has not automatically connected (and the error message « cannot connect to the server » has appeared), click on the lower left button of the application «Click to create TCP/IP connection» to reenable the connection.

>>> If the webots application still does not connect, reload the robota-webots application (or close and restart webots). Then, click again on the TCP/IP connection button of the Robota application.

>>> In case this still does not work and you keep receiving the error message, « cannot connect to the server », check that the windows applications of your PC (e.g. Microsoft Explorer) is set-up to communicate via TCP/IP.

> Type different motor commands in the upper right window and click on the « Click to send » button to let the simulated robot move. The motor commands take the form of the following :

« D, servo_id, position »

The servo_id takes value between 0 and 5. The position correspond to the angle in degree to which to send the motor. !!!Note that the motor commands for the simulator differ from that of the real robot. !!! Play around with these values to enable the simulated robot to move each of its five limbs back and forth.

TCP/IP FUNCTIONS :

This tutorial contains the same basic files as those presented in tutorial 1. The reader should be familiar with the basic Robota template of Section 1 to follow this tutorial. This template contains also two new files client.cpp and client.h,
that define functions and variables for the TCP/IP communication with the webots-application, as follows:

```
#define SOCKET_PORT 10020
#define SOCKET_SERVER "127.0.0.1" /* local host */
```

```c
int init_socket();
```
This function initializes the TCP/IP connection with the webots applet. It is called by `CRobotaDlg::OnInitDialog()`, i.e. when the application is launched, and by the function `CRobotaDlg::OnSocket()`, that is when the user clicks on the button *Click to create TCP/IP connection*.

```c
int interact_with_socket(char *message)
```
This function passes the parameter message to the webots applet through the TCP/IP connection. It is called by the function `CRobotaDlg::OnGo()`, i.e. when the user presses the button *Click to send*.

### 5.2 Tutorial 5: Webots-Robota Vision-Based Imitation game

In this tutorial, you will create an application in which the webots-robota simulation imitates the arm and head motion of user, similarly to what was done in Tutorial 3 of Section 3.

1. Update the template webots-Robota to add the function wintv.cpp and wintv.h of Tutorial 3 (10%).
2. Update the function tracking to send the correct motor commands to the simulator (40%).
3. Add the file Speech.cpp and Speech.ch. Create an introduction function that explains what the imitation game consists of. The function introduction siplays the explanation in a static window and launches the TTS engine, if a crosscheck button *sound* is checked on (30%).

Grading : 100%

The solution for this tutorial is given in the folder `Robota-client-Tutorial5/`

### 5.3 Tutorial 6: Webots-Robota Learning game

For this tutorial, starts from the applet given in `Robota-client-Tutorial5` directory. Launch the `Robota.dsw` project. Compile and execute the project (key F5). The following user interface will appear:
The lower right buttons of the camera implement the imitation game described in the Tutorial 3 of this manual. However, here the simulated Robota imitates the movements of your arms and head.

**Tutorial 6: EXERCISE:**

Create a learning routine that allows the Robota to learn a little dance taught by the user through the video-based imitation game. The applet should look as follows:

1. Create 3 buttons: « Teach » « Repeat » and « Reset », as shown below. (10%)
2. Create two new files RobotaLearn.cpp and RobotaLearn.h. Implement learning routines that allow a) to measure continuously the potentiometer value (place it in the RobotaDlg ::OnTimer function (5%), b) to detect a change of position of each limb nased on potentiometer reading (5%), c) that store the sequence of change of position in an array (alternatively, you can implement a time-delay neural network to store the sequence, and attach one neuron to each of the eight quadrans associated with each limb position) (40%), d) that can read back the sequence of stored position (or reactivate the neural network) (20%).

3. Update the functions OnTeach, Onrepeat and OnReset, so that these implement the following (20%):
   > Once the button teach has been pressed, the application starts recording the sequence of movements taught through the imitation game.
   > To stop the learning, the button teach must be pressed a second time.
   > Once the button repeat is pressed, the robot replays the sequence of movement.
   > During the teach and repeat parts of the game, a music file (rock.mp3) is being played through the winamp software, see Tutorial 1.

Grading : 100 %

Tutorial 6 : SOLUTION

The solution for tutorial 6 is given in the folder Robota-client-Tutorial6/. Below is a copy of the functions defined in the files RobotaLearn.cpp and RobotaLearn.h that implement the learning game.

/// DEFINE GLOBALE VARIABLES FOR LEARNING

Copyright: A.G. Billard, 2002
double state_sensor[Nsensors];
double state_global_sensor[Nsensors];

int pwm1,pwm2,pwm3,pwm4,pwm5,pwm6;
static double decay=0.85;
static int time_to_speak=0;

// Dance variable
int pt_dance, pt_in_dance;
int Wm[Nm_dance][Nm_steps];
int Wp[Nm_dance][Nm_steps];
long int Wt[Nm_dance][Nm_steps];

int pot[Nsensors];

// Variables of behaviour
int robot_state,music_choice,state_play, remember, dance_start;
static int mood[4]={-1,-1,-1,-1};
char robot_say[1];
static int mot_state=0;

float memory_length;    // Durée de la mémoire en secondes (float)

double F(double x)
{
    // if (x<0.00000001) return 0;
    if (x<0.02/(memory_length)) return 0;
    else if (x>=1)  return 1;
    else return x;
}

void init_memory()
int i,j;

for (i=0;i<Nsensors;i++) {
    state_sensor[i]=0;
}

pt_dance=0; pt_in_dance=0;
for (i=0;i<Nm_dance;i++) {
    for (j=0;j<Nm_steps;j++) {
        Wm[i][j]=0;
        Wp[i][j]=0;
        Wt[i][j]=0;
    }
}

.parseFloat

// Update neural functions
void update_sensor_state()
{
    int i;
    // Update sensor states
    for (i=0;i<Nsensors;i++){
        state_sensor[i]=F(state_sensor[i]*decay);
        state_global_sensor[i]=F(state_global_sensor[i]*decay);
    }
}

// Routines for storing and repeating sequence of movement (dance pattern)
void learn_dance(int N, int z){
    int i=pt_dance;
    int j=pt_in_dance;
    if ((j==0) || (fabs(z-pot[N])>2)) {
        Wm[i][j]=N; Wp[i][j]=z; Wt[i][j]=clock();
        pot[N]=z;
    }
}
pt_in_dance++;
}

void repeat_dance(){
    char message_1[1040];
    int i=pt_dance;
    int j=1;
    zero_state(0);
    if (Wm[i][j]>0){
        int T=clock();
        do {
            if (Wt[i][j]-Wt[i][j-1]>0) while ((clock()-T)<(Wt[i][j]-Wt[i][j-1]));
            int N=Wm[i][j];
            if (N==2) sprintf(message_1,"5a%dc",Wp[i][j]);
            else if (N==3) sprintf(message_1,"5e%dg",Wp[i][j]);
            else if (N==4) sprintf(message_1,"5m%dk",Wp[i][j]);
            else if (N==5) sprintf(message_1,"5i%do",Wp[i][j]);
            else if (N==1) sprintf(message_1,"5q%ds",Wp[i][j]);
            else if (N==6) sprintf(message_1,"5q%ds",Wp[i][j]);
            else sprintf(message_1,"b");
            send_order_to_robot(hCom,message_1);
            T=clock();
            j++;
        } while ((j<Nm_steps) & (Wm[i][j]>0) & (Wt[i][j]-Wt[i][j-1]>=0));
        zero_state(0);
    }
6 PocketPC – Version of Robota

This section requires that you have purchased a PocketPC, running Windows CE. All tutorials of this section are known to work on a HP-Compaq IPAQ 3850.

6.1 PocketPC : Guideline for Software Installation

Go to the folder PocketPC/System/ of the CD-rom and install all the systems programs required to run the PocketPC and its emulator.

6.1.1 Installation of the programming environment, required for all tutorials:

1. Install the Microsoft's Embedded Visual Tools, which you can download freely from the Microsoft website http://www.microsoft.com/mobile/downloads/emvt30.asp (note that all these URL may change location).

   This provides you with an environment with which you will develop the Robota applications for the Pocket PC. The Embedded Visual Tools environment is similar to the Visual C++ environment. However, a number of binaries differ and code written in Visual C++ does not port directly to Embedded Visual C++.

2. Install the Pocket PC 2002 SDK libraries by downloading the HPC2K-SDK_USA libraries from the Microsoft website. This will give you a set of sample applications for the PocketPC.

3. Download and Install the Microsoft's PocketPC 2002 Emulator Images, see http://www.microsoft.com/mobile/developer/downloads/default.asp This provides you with an emulator of Pocket PC, with which you can start developing your application before purchasing the PocketPC. Click on the link Pocket PC  more from the above website.

4. Install the ActiveSync libraries located in the directory PocketPC/System/ActiveSync/ of the RobotaUserGuide CD-rom, This software allows to synchronize the PC and the PocketPC. When installing the software, plug the PocketPC to one of the USB entries of the PC.
6.1.2 Installations required for Tutorial 8:


6.1.3 Installations required Tutorial 9:

1. Go to the folder PocketPC/Speech/. Install the libraries for the Conversay speech engine (recognition and synthesizes of speech) from the directory CONVERSAY/. Look at the Installation file for step-by-step instructions. Conversay is a speech engine that provides both recognition and synthesis. Conversay synthesis provides only an English Male Voice. See http://www.conversay.com/Products/Embedded/MC.asp

2. Install the libraries for the ELAN synthesizer from the directory ELAN/. ELAN is a text-to-speech engine that provides female and male voices in different languages: French, English, German, etc. See www.elan.fr Look at the Installation file for instructions!

6.1.4 Basic Files of the RobotaPocketPC Workspace:

Go to the folder PocketPC/Tutorials/Tutorial7/. Copy the folder Tutorial/ locally on your hard drive. Launch the workspace RobotaPPC by double clicking on RobotaPPC.vcw. The workspace looks as follows:
Robota PocketPC Template Files

Source files
Header files
Resource files
External dependencies

Source files:
RobotaPPC.rc: Graphical Resources (see explanation below).
RobotaPPCDialog.cpp: Functions for interacting with the Dialog box.
SerialPort.cpp: Functions for the serial port connection
RobotaPPCMotor.cpp: Function for interacting with Robota’s motors and sensors
StdAfx.cpp: Includes just the standard #includes.

Header files:
Resource.h: Microsoft Developer Studio generated include file.
RobotaPPCDialog.h: Header file for the RobotaPPCDialog.cpp file.
SerialPort.h: Header file for the SerialPort.cpp file
StdAfx.h: File for standard system includes files.

Robota Resources
Bitmap
Dialog (Here is the Dialog Box)
String Table
Version

Robota Classes
BasicDialog
Globals

RobotaPPCMotor.cpp contains general functions for handling strings, functions to initialize the serial port, functions to send motor commands to the Motor Board and functions to send requests to the Motor Board and Sensor Board to read the status of the potentiometers and of the body switches. These are the same functions as those defined in RobotMotors.cpp, in the PC version of Robota, see Section 3.1

RobotaPPCDialog.cpp contains functions to manage the dialogue interface on the PC.

int WINAPI WinMain
This function contains general initialization function. It is called once when the application is launched. You should add your initialization functions here.

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) This function launches the main dialog window. It refreshes the screen and destroys the dialog window when the exit signal is being sent.

Copyright: A.G. Billard, 2002
!! **BEWARE** that it only destroys the dialog window. If you launch other tasks in parallel (callback function, multi-threading), such as grabbing the image, grabbing the sound of the microphone, etc, make sure that you kill those threads when receiving the exit signal. Otherwise, they will continue running in the background and might prevent you from relaunching the program.

LRESULT CALLBACK BasicDlgProc(HWND hwnd, UINT msg, WPARAM wp, LPARAM lp)
This callback function handles the various messages sent from the dialog window.

*SerialPort.cpp* contains functions to manage the serial port.

**PortInitialize()** This function initializes the serial port. It returns 0 if the serial port is not responding or had already been open. Make sure that the parameters (COM Port number, baudrate, etc) are correctly set-up to communicate with Robota Motor Board, see Section 2.1. Make sure that your program calls this function before trying to read or write on the port!

**PortWrite** and **PortReadThread** write and read data through the serial port.

**PortClose** This function closes the serial port. Make sure that it is being called before closing the application!

### 6.1.5 To Compile and Download Files to PocketPC and Emulator

In order to download the application in the emulator, select the option **Win32(WCE X86) Debug** and **PocketPC 2002 Emulation**, see below.

In order to download the application on the PocketPC, select the option **Win32(WCE ARM) Debug** and **PocketPC 2002 (Default Device)**, see below.

Copyright: A.G. Billard. 2002
Compile (Key F7) or BUILD ➔ Rebuild All and then download (Ctr + F5), the application will start downloading to the emulator or PocketPC and will display the following:

Once the download has been completed, you should see the dialogue window pop up on the screen of the emulator or PocketPC. If this does not appear, launch the application manually from the PocketPC or Emularo: click on Start ➔ RobotaPPC

### 6.2 Tutorial 7 – Serial Port Communication

Copy the Embedded C++ template for this tutorial from the folder Tutorial7/Tutorial. Complete the following steps:

1. Update the dialogue box in the resource file RobotaPPC.rc to create two new buttons MOVE_ARM and MOVE_LEG (40%).

2. Update the BasicDlgProc function in RobotaPPCDialog.cpp, to respond to a mouse click on each of the two buttons MOVE_ARM and MOVE_LEG by an appropriate motion of Robota’s leg and arm (60%).

**Grading:** 100 points, divided following the above percentages.

Once you have completed all steps, compare your C++ functions to that given in the folder Tutorial7/Solution/. Launch the application RobotaPPC.exe from the Release directory. The Dialog box for this tutorial should look like this:
6.3 Tutorial 8 – Camera: Robota Imitates

This tutorial requires a Camera. The software for this tutorial is known to work with the FlyCAM-CF: see section 6.1.2 for instructions on how to install the camera. This tutorial assumes that you have done Tutorial3 for the PC version of Robota, see Section 3.3.

Copy the visual C++ template for this tutorial from the folder Tutorial8/Tutorial/ in the Tutorial CD-Rom. Copy the images located in the folder Tutorial8/Tutorial/images into the PocketPC folder Robota/ (Use the explore function of the ActiveSync software to locate the Robota folder on the PocketPC; then drag and drop the images from the PC folder Tutorial8/Tutorial/images to the PocketPC Robota folder).

This tutorial implements a tracking system, similar to the one described in Tutorial 3, see Section 3.4. It lets the robot tracks and imitate in mirror fashion the movements of the left arm and head (more precisely of the nose) of the user. The dialogue box looks like this:
The template contains the files VisionSystem.cpp and VisionSystem.h, that were not included in the basic template of Tutorial 6, see Section 6.1.1. VisionSystem.cpp contain functions to display and read the camera input. VisionSystem.h contains header for the file wintv.cpp. The function of VisionSystem.cpp relevant for this tutorial is:

```c
void rotate90((unsigned char *)buffercam2, (unsigned char *)buffercam);
```
This function rotates the image returned by the camera in the pointer buffercam and store the image in a new buffer (buffercam2). All further processing is done on buffercam2.

**Tutorial 8: Exercise**

Complete the following steps: **Grading**: 100 points.

1. Update the different functions of the file VisionSystem.cpp to track the motions of the right arm, by measuring the variation of brightness in the three areas of the right hand-side of the image. The camera pixels are stored in an array called hPixel. Each camera pixel corresponds to 3 RGB hPixel values (RGB stands for red-green-blue). Look at the MotionTracking function to find out how to read and interpret the values of the camera pixels (60%).

2. Let the robot move in mirror fashion to the user: i.e. the robot’s left arm moves when the user’s right arm moves (20%).
6.4 Tutorial 9 – Camera : Robota Speaks

This tutorial requires the CONVERSAY speech engine, that provides both Automatic Speech Recognition (ASR) and Text-To-Speech synthesisis (TTS), and the ELAN TTS engine. See section 6.1.3 for instructions on how to install the speech engines components.

Copy the visual C++ template for this tutorial from the folder Tutorial9/Tutorial/ in the Tutorial CD-Rom of the Robota Kit. Copy the images located in the folder Tutorial9/Tutorial/images into the PocketPC folder Robota/ (Use the explore function of the ActiveSync software to locate the Robota folder on the PocketPC; then, drag and drop the images from the PC folder Tutorial8/Tutorial/images to the PocketPC Robota folder). This tutorial implements a simple dialogue system in which Robota responds “hello you”, when you say “hello”. The dialogue box looks like this:
Click on the **“RUN SR”** or the **“STOP SR”** buttons to launch or stop respectively the CONVERSAY and ELAN speech engines.

The template contains the files `speechrecognition.cpp`, `speechrecognition.h`, `speechsynthesis.cpp`, `speechsynthesis.h`, that were not included in the basic template of Tutorial 6, see Section 6.1.1.

**speechrecognition.cpp** contains:

- `int RunSR()` Initialize and run Conversay speech recognition API
- `int StopSR()` Stop the Conversay speech recognition API
- `void Callback(const EventMsg* pEventMsg, void* pvUserData, void* pv)` This callback function detects if a word stored in the grammar has been detected, and, if it the case, launches the ELAN TTS engine to answer.

**speechsynthesis.cpp** contains:

- `BOOL initElan()` Initialize the Elan Speech Synthesis API
- `void closeElan()` Release the resources of the Elan Speech Synthesis API
- `void speakElan(TCHAR *cText)` Takes as input a text string, and transforms it into speech

**Tutorial 8: Exercise**

Update the callback function of the file file `speechrecognition.cpp` to create a little dialogue between Robota and you.

**6.5 SAMPLE CODE**

The folders `/samplecode/Imitation Learning using Speech and Potentiometers` and `/samplecode/Imitation Learning using Speech and Vision` provide two samples of applications, that combine the imitation game of Tutorial8 and the language application of Tutorial9 to implement a game, in which the robot can be taught a simple language through imitation.