Internal Report 2010-03 January 2010

Universiteit Leiden

Opleiding Informatica

Camera Controlled Robot

Johan IJsveld

BACHELORSCRIPTIE

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

This project involves a windows program that can detect human movement and gestures in a video. There is also a robot that has a built in camera.

The aim is to use the algorithm from the windows program to process pictures from the robot and to use the detected gestures to control the robot.

Contents

1 Introduction

In this project a robot is controlled using its camera. The images from the camera are put through an existing algorithm that can detect human movements and gestures. These movements are then mimiced by the robot.

The algorithm detects the following movements:

- Left hand movement;
- Right hand movement;
- Body turning left;
- Body turning right;
- Body moving up;
- Body moving down.

To keep the processing needs of the robot down, the algorithm is run in a seperate program running on a PC. This PC connects to the robot and receives pictures over a wireless connection. It then sends back, to the robot, any gestures or movements found. The robot will then react to these gestures.

2 Implementation

The project is split into two main parts. First, the robot running its own program implementing its behaviour. Second, a program running on a PC that communicates with the robot and runs the algorithm that recognizes the gestures.

2.1 Robot

The robot that is used, is the Sony Aibo ESR-7[3]. The robot is running Urbi[1], which is the OS of the robot. Urbi also contains a interpreter with its own language[6].

Urbi also allows connections to be made to it over TCP/IP, e.g. with telnet, to send commands or programs to it. The robot has a built-in Wi-Fi card.

The Aibo robot is further described in section 4.

2.2 Program

The program running on the PC is C++. This program sends the Urbi code to the robot that implements the behaviour of the robot. It also receives information and pictures from the robot over the wireless connection.

The graphics library SDL[7] version 1.2.13 is used to display the camera pictures, movements detected and other information from the robot. Also SDL_ttf[8] version 2.0.9 is used to display text. The library libUrbi[5] is used for connecting to the robot. Version 1.5 of Urbi is used[6].

The algorithm that recognizes gestures is a static library and is implemented in C. This algorithm has been extracted and cleaned up from a Windows program.

The program is further described in section 5 and the Urbi program in section 6.

3 Theory

Figure 1 shows the state diagram that describes the behaviour of the Aibo robot. To keep the number of state transitions down, all movements go through one of the main postures: laying, sitting and standing. From there the final movement is made.

To go to an other gesture, the old gesture is undone by going back to the main posture.

One extra state that does not correspond to any gestures is added. This is the idle state. The Aibo robot will return to this state whenever there is a period of inactivity.

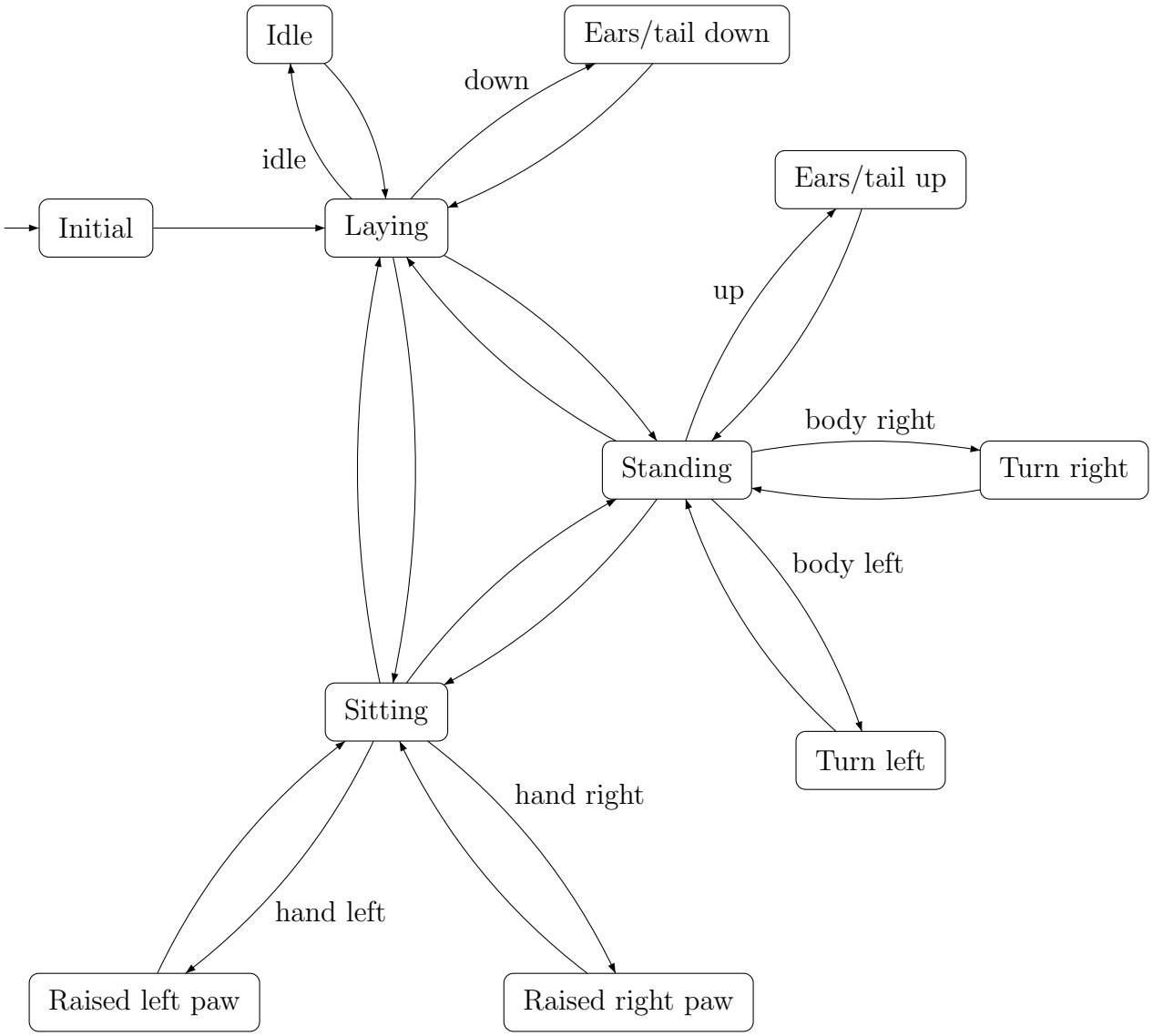

Figure 1: The state diagram of the Aibo robot.

4 The Aibo robot

4.1 Sensors

The Aibo robot has the following sensors:

- Distance sensors in the nose and chest;
- Touch sensors on the head, back and paws;
- A camera in the nose.

The camera in the nose is the only sensor that is used for this project.

4.2 Motors

The robot has many motors. It can move all its four legs at three joints. It can tilt and pan its head and it can move its neck, tail and ears. All of these possible movements are used in mimicing the gestures.

4.3 Leds and sounds

There are leds on the face, top of the head and back. The leds on the back glow to show urbi is running. The face leds are used to show eyes.

The leds on the head show the status of the robot. Green, when the robot is waiting and ready to start mimicing gestures. Red, to show it is busy mimicing a new gesture. The Aibo robot will ignore any new detected gestures while it is busy.

The Aibo robot barks two times when it has detected a new gesture and is followed by one loud when the detected gesture is ignored.

5 The Program

5.1 Connection to the Aibo robot

The connection to the robot is made using the Urbi library. This is done in the Urbi class, in the files `urbi.hpp` and `urbi.cpp`. See listings 2 and 3.

5.1.1 Receiving messages

After connecting to the Aibo robot a number off callbacks is set up, to receive messages from the robot. The callbacks used are listed in `callbacks.hpp`, see listing 4.

Associated with each callback is a tag. These tags can be used in the urbi script files to send data to the correct callback.

Each callback also gets a pointer to a structure with data, defined in `callbackdata.hpp`, see listing 6. This contains pointers to the screen to update information and to the client object to send commands to the robot.

5.1.2 Sending the Urbi program

The entire urbi program is also sent in the Urbi class to the robot. It is sent, instead of written directly on the robot's memory stick to make development faster. All the urbi files that need to be sent are read from a seperate file in the scripts directory: `scripts.lst`, listing 16. The urbi scripts are sent in the order that they appear in the file.

5.1.3 Sending gesture events

The program receives messages with camera data in the callback `DisplayImage()`. Defined in `callbacks.cpp`, see listing 5.

This callback will then convert the image from a jpeg file to a raw image and feeds this to the algorithm. Then it updates the screen to display it with the result. The call to the algorithm is declared in `transform.h`, see listing 33.

When the algorithm finds a gesture in the image data, it will call a new callback SetGesture(), also defined in `callbacks.cpp`, that sends the appropriate event to the robot. The robot can then respond to that event.

5.2 Usage

5.2.1 Compiling

To compile the gesture library simply run `make`.

To compile the aibo program. Use either `Makefile.mingw32` for compiling under windows with mingw32 or `Makefile.linux` for compiling under Linux with gcc. Make sure that the paths to the `gesture`, `SDL`, `SDL_ttf`

and `libUrbi` libraries are correct in the makefile. Under linux the program `sdl-config` should supply the correct flags to gcc for the SDL library.

Then run either `make -f Makefile.mingw32` or `make -f Makefile.linux`.

5.2.2 Running

To run the aibo program. Turn the Aibo robot on and let Urbi startup and make the Wi-Fi connection. Type `aibo [address]` in the `bin` directory. Where addres is an optional ip address of the Aibo robot. The default address is 192.168.2.3.

Once started the program should make a connection to the Aibo robot and send the urbi program. The robot should start to initialise itself. The program will then begin showing images and status information from the robot.

5.2.3 Keys

The following keys can be used while running the aibo program:

Key	Description
Tab	Switch on/off the sending of detected gesture events.
Arrow up	Scroll log text up.
Arrow down	Scroll log text down.
F1	Simulate hand left event.
F2	Simulate hand right event.
F3	Simulate body left event.
F4	Simulate body right event.
F5	Simulate up event.
F6	Simulate down event.
F7	Simulate idle event.
F12	Run the simulation from the `simulate.u` script.
Esc	Resend all the scripts.

6 The Urbi program

6.1 Support code

The scripts `eyes.u`, `bark.u`, `paws.u`, `tail.u` and `ears.u` define classes to implement some basic behaviour. While `support.u` defines objects using

these classes and some events and functions to make using them easier.

6.2 Setting up

The script `camera.u` sets up the camera and the loop that keeps sending pictures. `loops.u` creates message loops for various status messages, i.e. the cpu load, free memory and remaining battery power. `setup.u` turns on the motors and puts the Aibo robot in its initial state.

6.3 The behaviour

The scripts `hands.u`, `body.u`, `up.u`, `down.u`, `idle.u` define the actions for the various gestures that can be detected. They contain the transitions to the various states and the reverse transitions to get back to the basic posture. As shown in figure 1.

In `events.u` the responses to the gesture events received from the PC are handled, using the transition functions described above. It also keeps track of the state the Aibo is currently in and updates it.

7 Conclusion

By using Urbi it was possible to communicate between the PC and robot. The Aibo robot sends pictures to be analyzed by the algorithm on the PC. The PC sends the results back to the robot and the robot reacts to them by mimicing the events.

The used algorithm though, is not very accurate and very sensitive to light and background conditions. It was not possible to consistently detect the correct gestures.

A future possibilty to extend this project is then to use a different algorithm for detection of gestures.

References

[1] Website Urbi
http://www.urbiforge.com/
[June 24, 2009]

[2] Urbi 1.5 for Aibo
http://www.gostai.com/aibo.html
[June 24, 2009]

[3] Aibo ERS-7 documentation for Urbi
http://www.gostai.com/doc/en/aibo/aibo.ers7.html
[June 24, 2009]

[4] Urbi tutorial
http://www.gostai.com/doc/en/urbi-tutorial-1.5/index.html
[June 24, 2009]

[5] libUrbi
http://www.gostai.com/doc/en/liburbi-1.0/
[June 24, 2009]

[6] Urbi specification
http://www.gostai.com/doc/URBI-Specif-1.5.pdf
[June 24, 2009]

[7] Website SDL
http://www.libsdl.org/
[June 24, 2009]

[8] Website SDL_ttf
http://www.libsdl.org/projects/SDL_ttf/
[June 24, 2009]

A C++ program

A.1 aibo.cpp

```
1  #include "SDL.h"
2
3  #include "urbi.hpp"
4  #include "events.hpp"
5  #include "screen.hpp"
6
7  #include <iostream>
8  #include <cassert>
9
10 using namespace std;
11 using namespace Aibo;
12
13 const char * const default_address = "192.168.2.3";
14
15 int main(int argc, char *argv[])
16 {
17     const char *address = default_address;
18     if (argc > 1) {
19         address = argv[1];
20     }
21
22     try {
23         Screen screen(640, 480, 24);
24         Urbi urbi(address, screen);
25         Events events(urbi.GetCBData());
26         events.Process();
27     }
28     catch (bad_alloc& ba) {
29         cerr << "Out of memory." << endl;
30     }
31     catch (...) {
32         cerr << "Unhandled exception." << endl;
33     }
34
35     UrbiExit();
36     return 0;
37 }
```

Listing 1: aibo.cpp

A.2 urbi.hpp

```
1  #ifndef URBI_H
2  #define URBI_H
3
4  #include "screen.hpp"
5  #include "callbackdata.hpp"
6  #include "timer.hpp"
7
8  #include "urbi/fwd.hh"
9  #include "urbi/uabstractclient.hh"
10
11 #include <string>
12
13 namespace Aibo {
14
15 // Handles the connection and urbi scripts.
16 class Urbi {
17 public:
18     Urbi(const std::string& address, Screen& screen);
19     ~Urbi();
20
21     urbi::UClient& Client() const;
22     CallbackData *GetCBData();
23
24     void ResendScripts();
```

```
25  private:
26      static const char * const script_location;
27      static const char * const scripts_list;
28      void SetCallbacks();
29      void SendScripts(const char *directory, const char *list_filename, urbi::UClient&
            client);
30
31      std::string aibo_address;
32      urbi::UClient& client;
33      Timer timer;
34      CallbackData cbdata;
35
36      std::vector<urbi::UCallbackID> callback_ids;
37  };
38
39  // Cleanly exits the program for Urbi.
40  void UrbiExit();
41
42  } // namespace Aibo
43
44  #endif /* URBI_H */
```

Listing 2: urbi.hpp

A.3 urbi.cpp

```
1   #include "urbi.hpp"
2   #include "callbacks.hpp"
3   #include "callbackdata.hpp"
4
5   #include "urbi/uabstractclient.hh"
6   #include "urbi/uclient.hh"
7   #include "urbi/ubinary.hh"
8   #include "urbi/ublend-type.hh"
9   #include "urbi/ucallbacks.hh"
10  #include "urbi/uconversion.hh"
11  #include "urbi/uexternal.hh"
12  #include "urbi/uobject.hh"
13  #include "urbi/uproperty.hh"
14  #include "urbi/ustarter.hh"
15  #include "urbi/usystem.hh"
16  #include "urbi/utag.hh"
17  #include "urbi/uvalue.hh"
18  #include "urbi/uvar.hh"
19
20  #include <string>
21  #include <iostream>
22  #include <fstream>
23
24  using namespace std;
25  using namespace urbi;
26  using namespace Aibo;
27
28  const char * const Urbi::script_location = "../scripts/";
29  const char * const Urbi::scripts_list = "scripts.lst";
30
31  Urbi::Urbi(const string& address, Screen& screen)
32      : aibo_address(address),
33        client(urbi::connect(aibo_address.c_str())),
34        timer(1000, 0),
35        cbdata(&screen, this, &client)
36  {
37      // check if we are connected to aibo
38      if (client.error() != 0) {
39          std::cout << "Error connecting to: " << aibo_address << std::endl;
40          urbi::exit(1);
41      } else {
42          std::cout << "Connected to: " << aibo_address << std::endl;
43      }
44
45      SetCallbacks();
46      SendScripts(script_location, scripts_list, client);
47  }
48
```

```
49  Urbi::~Urbi()
50  {
51      // remove callback functions
52      for (unsigned int i = 0; i < callback_ids.size(); ++i) {
53          client.deleteCallback(callback_ids[i]);
54      }
55  }
56
57  urbi::UClient& Urbi::Client() const
58  {
59      return client;
60  }
61
62  CallbackData *Urbi::GetCBData()
63  {
64      return &cbdata;
65  }
66
67  void Urbi::ResendScripts()
68  {
69      //client.send("reset;");
70      client.send("log << echo(\"Resending scripts\");");
71      SendScripts(script_location, scripts_list, client);
72  }
73
74  void Urbi::SetCallbacks()
75  {
76      callback_ids.push_back( client.setCallback(LogMessage, (void *)"log.txt", "") );
77      callback_ids.push_back( client.setWildcardCallback(callback(LogMessage, (void *)"log
            .txt")) );
78      //callback_ids.push_back( client.setErrorCallback(callback(LogMessage, (void *)("
            errors.txt"))) );
79      //callback_ids.push_back( client.setCallback(LogMessage, (void *)("warnings.txt"), "
            warning") );
80
81      callback_ids.push_back( client.setErrorCallback(callback(PrintMessage, &cbdata)) );
82      callback_ids.push_back( client.setCallback(PrintMessage, &cbdata, "warning") );
83
84      callback_ids.push_back( client.setCallback(PrintMessage, &cbdata, "log") );
85      callback_ids.push_back( client.setCallback(PrintMessage, &cbdata, "start") );
86      callback_ids.push_back( client.setCallback(PrintMessage, &cbdata, "ident") );
87      callback_ids.push_back( client.setCallback(PrintMessage, &cbdata, "setup") );
88      callback_ids.push_back( client.setCallback(DisplayImage, &cbdata, "image") );
89      callback_ids.push_back( client.setCallback(SetCPULoad, &cbdata, "load") );
90      callback_ids.push_back( client.setCallback(SetFreeMemory, &cbdata, "memory") );
91      callback_ids.push_back( client.setCallback(SetPowerLevel, &cbdata, "powerlevel") );
92      callback_ids.push_back( client.setCallback(SetLatency, &cbdata, "latency") );
93      callback_ids.push_back( client.setCallback(PrintMessage, &cbdata, "gesture") );
94  }
95
96  void Urbi::SendScripts(const char *directory, const char *list_filename, UClient& client
        )
97  {
98      string filename = string(directory) + string(list_filename);
99      ifstream scripts_file(filename.c_str());
100
101     const int size = 255;
102     char line[size];
103
104     while (scripts_file.getline(line, size)) {
105         string script_filename(line);
106         if (!script_filename.empty() && script_filename[0] != '#') {
107             filename = string(directory) + string(script_filename);
108             cout << "Sending \'" << filename << "\'." << endl;
109             client.sendFile(filename.c_str());
110         }
111     }
112 }
113
114 void Aibo::UrbiExit()
115 {
116     urbi::exit(1);
117 }
```

Listing 3: urbi.cpp

A.4 callbacks.hpp

```
#ifndef CALLBACKS_HPP
#define CALLBACKS_HPP

#include "urbi/fwd.hh"
#include "urbi/uabstractclient.hh"
#include "urbi/uclient.hh"
#include "urbi/ucallbacks.hh"

namespace Aibo {

urbi::UCallbackAction PrintMessage(void *data, const urbi::UMessage& msg);
urbi::UCallbackAction LogMessage(void *data, const urbi::UMessage& msg);
urbi::UCallbackAction DisplayImage(void *data, const urbi::UMessage& msg);
urbi::UCallbackAction SetCPULoad(void *data, const urbi::UMessage& msg);
urbi::UCallbackAction SetFreeMemory(void *data, const urbi::UMessage& msg);
urbi::UCallbackAction SetPowerLevel(void *data, const urbi::UMessage& msg);
urbi::UCallbackAction SetLatency(void *data, const urbi::UMessage& msg);

} // namespace Aibo

#endif /* CALLBACKS_HPP */
```

Listing 4: callbacks.hpp

A.5 callbacks.cpp

```
#include "callbacks.hpp"
#include "callbackdata.hpp"
#include "urbi_stream.hpp"
#include "screen.hpp"

#include "transform.h"

#include "urbi/uabstractclient.hh"
#include "urbi/uclient.hh"
#include "urbi/ubinary.hh"
#include "urbi/ublend-type.hh"
#include "urbi/ucallbacks.hh"
#include "urbi/uconversion.hh"
#include "urbi/uexternal.hh"
#include "urbi/uobject.hh"
#include "urbi/uproperty.hh"
#include "urbi/ustarter.hh"
#include "urbi/usystem.hh"
#include "urbi/utag.hh"
#include "urbi/uvalue.hh"
#include "urbi/uvar.hh"

#include "SDL.h"

#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <cassert>

using namespace std;
using namespace urbi;
using namespace Gestures;
using namespace Aibo;

namespace Aibo {
static void SetGesture(gestures_t gesture, void *data)
{
    CallbackData *cbdata = static_cast<CallbackData *>(data);
    Screen *screen = cbdata->screen;
    UClient *client = cbdata->client;
    bool send_events = cbdata->send_events;

    string str;
    switch (gesture) {
```

```
46          case gesture_none:
47              break;
48          case gesture_body_left:
49              str = "bodyleft";
50              break;
51          case gesture_body_right:
52              str = "bodyright";
53              break;
54          case gesture_hand_left:
55              str = "handleft";
56              break;
57          case gesture_hand_right:
58              str = "handright";
59              break;
60          case gesture_up:
61              str = "up";
62              break;
63          case gesture_down:
64              str = "down";
65              break;
66      }
67
68      screen->SetGesture(GestureString(gesture));
69      if (send_events) {
70          client->send("emit %s;", str.c_str());
71      }
72      cerr << "emit " << str << ";" << endl;
73  }
74  } // namespace Aibo
75
76  UCallbackAction Aibo::PrintMessage(void *data, const UMessage &msg)
77  {
78      CallbackData *cbdata = static_cast<CallbackData *>(data);
79      Screen *screen = cbdata->screen;
80      //UClient *client = cbdata->client;
81
82      stringstream ss;
83      StreamMessage(ss, msg);
84
85      screen->PrintLine(ss.str());
86      return URBI_CONTINUE;
87  }
88
89  UCallbackAction Aibo::LogMessage(void *data, const UMessage& msg)
90  {
91      const char *filename = static_cast<const char *>(data);
92      ofstream ofs(filename);
93      StreamMessage(ofs, msg);
94      return URBI_CONTINUE;
95  }
96
97  UCallbackAction Aibo::DisplayImage(void *data, const UMessage &msg)
98  {
99      CallbackData *cbdata = static_cast<CallbackData *>(data);
100     Screen *screen = cbdata->screen;
101     //UClient *client = cbdata->client;
102
103     unsigned char buffer[500000];
104     int sz = 500000;
105
106     std::cout << "Image: " << msg.timestamp << " \'" << msg.tag << "\'" << std::endl;
107     switch (msg.type) {
108         case urbi::MESSAGE_SYSTEM:
109             std::cout << msg.message << std::endl;
110             break;
111         case urbi::MESSAGE_ERROR:
112             std::cout << msg.message << std::endl;
113             break;
114         case urbi::MESSAGE_DATA:
115             urbi::UValue& value = *(msg.value);
116             urbi::UBinary& binary = *(value.binary);
117             assert(binary.type == urbi::BINARY_IMAGE);
118             urbi::UImage& image = binary.image;
119             std::cout << "size: " << image.size
120                       << ", width: " << image.width
121                       << ", height: " << image.height
122                       << ", format: " << image.format_string()
```

```
123                         << std::endl;
124
125             urbi::convertJPEGtoRGB(image.data, image.size, buffer, sz);
126             Gestures::Transform(buffer, image.width, image.height, 3*image.width*image.
                    height, SetGesture, cbdata);
127             screen->SetImage(buffer, image.width, image.height);
128             break;
129     }
130
131     return URBI_CONTINUE;
132 }
133
134 UCallbackAction Aibo::SetCPULoad(void *data, const UMessage &msg)
135 {
136     CallbackData *cbdata = static_cast<CallbackData *>(data);
137     Screen *screen = cbdata->screen;
138     //UClient *client = cbdata->client;
139
140     std::cout << "CPU load: " << msg.timestamp << " \'" << msg.tag << "\'" << std::endl;
141     switch (msg.type) {
142         case urbi::MESSAGE_SYSTEM:
143             std::cout << msg.message << std::endl;
144             break;
145         case urbi::MESSAGE_ERROR:
146             std::cout << msg.message << std::endl;
147             break;
148         case urbi::MESSAGE_DATA:
149             urbi::UValue& value = *(msg.value);
150             switch (value.type) {
151                 case urbi::DATA_DOUBLE:
152                     std::cout << "Double: " << value.val << std::endl;
153                     screen->SetCPULoad(value.val);
154                     break;
155                 case urbi::DATA_STRING:
156                     std::cout << "String: " << *(value.stringValue) << std::endl;
157                     break;
158                 case urbi::DATA_BINARY:
159                     std::cout << "Binary: " << value.binary << std::endl;
160                     break;
161                 case urbi::DATA_LIST:
162                     std::cout << "List: " << value.list << std::endl;
163                     break;
164                 case urbi::DATA_OBJECT:
165                     std::cout << "Object: " << value.object << std::endl;
166                     break;
167                 case urbi::DATA_VOID:
168                     std::cout << "Void: " << value.storage << std::endl;
169                     break;
170             }
171     }
172
173     return URBI_CONTINUE;
174 }
175
176 UCallbackAction Aibo::SetFreeMemory(void *data, const UMessage& msg)
177 {
178     CallbackData *cbdata = static_cast<CallbackData *>(data);
179     Screen *screen = cbdata->screen;
180     //UClient *client = cbdata->client;
181
182     std::cout << "Free memory: " << msg.timestamp << " \'" << msg.tag << "\'" << std::
            endl;
183     switch (msg.type) {
184         case urbi::MESSAGE_SYSTEM:
185             std::cout << msg.message << std::endl;
186             break;
187         case urbi::MESSAGE_ERROR:
188             std::cout << msg.message << std::endl;
189             break;
190         case urbi::MESSAGE_DATA:
191             urbi::UValue& value = *(msg.value);
192             switch (value.type) {
193                 case urbi::DATA_DOUBLE:
194                     std::cout << "Double: " << value.val << std::endl;
195                     screen->SetFreeMemory(value.val);
196                     break;
197                 case urbi::DATA_STRING:
```

```
198                         std::cout << "String: " << *(value.stringValue) << std::endl;
199                         break;
200                     case urbi::DATA_BINARY:
201                         std::cout << "Binary: " << value.binary << std::endl;
202                         break;
203                     case urbi::DATA_LIST:
204                         std::cout << "List: " << value.list << std::endl;
205                         break;
206                     case urbi::DATA_OBJECT:
207                         std::cout << "Object: " << value.object << std::endl;
208                         break;
209                     case urbi::DATA_VOID:
210                         std::cout << "Void: " << value.storage << std::endl;
211                         break;
212                 }
213         }
214 
215         return URBI_CONTINUE;
216     }
217 
218     UCallbackAction Aibo::SetPowerLevel(void *data, const UMessage &msg)
219     {
220         CallbackData *cbdata = static_cast<CallbackData *>(data);
221         Screen *screen = cbdata->screen;
222         //UClient *client = cbdata->client;
223 
224         std::cout << "Power level: " << msg.timestamp << " \'" << msg.tag << "\'" << std::
              endl;
225         switch (msg.type) {
226             case urbi::MESSAGE_SYSTEM:
227                 std::cout << msg.message << std::endl;
228                 break;
229             case urbi::MESSAGE_ERROR:
230                 std::cout << msg.message << std::endl;
231                 break;
232             case urbi::MESSAGE_DATA:
233                 urbi::UValue& value = *(msg.value);
234                 switch (value.type) {
235                     case urbi::DATA_DOUBLE:
236                         std::cout << "Double: " << value.val << std::endl;
237                         screen->SetPowerLevel(value.val);
238                         break;
239                     case urbi::DATA_STRING:
240                         std::cout << "String: " << *(value.stringValue) << std::endl;
241                         break;
242                     case urbi::DATA_BINARY:
243                         std::cout << "Binary: " << value.binary << std::endl;
244                         break;
245                     case urbi::DATA_LIST:
246                         std::cout << "List: " << value.list << std::endl;
247                         break;
248                     case urbi::DATA_OBJECT:
249                         std::cout << "Object: " << value.object << std::endl;
250                         break;
251                     case urbi::DATA_VOID:
252                         std::cout << "Void: " << value.storage << std::endl;
253                         break;
254                 }
255         }
256 
257         return URBI_CONTINUE;
258     }
259 
260     UCallbackAction Aibo::SetLatency(void *data, const UMessage& msg)
261     {
262         CallbackData *cbdata = static_cast<CallbackData *>(data);
263         Screen *screen = cbdata->screen;
264         //UClient *client = cbdata->client;
265 
266         std::cout << "Latency: " << msg.timestamp << " \'" << msg.tag << "\'" << std::endl;
267         switch (msg.type) {
268             case urbi::MESSAGE_SYSTEM:
269                 std::cout << msg.message << std::endl;
270                 break;
271             case urbi::MESSAGE_ERROR:
272                 std::cout << msg.message << std::endl;
273                 break;
```

```
274             case urbi::MESSAGE_DATA:
275                 urbi::UValue& value = *(msg.value);
276                 switch (value.type) {
277                     case urbi::DATA_DOUBLE:
278                         {
279                             std::cout << "Double: " << value.val << std::endl;
280                             double tmp = value.val;
281                             Uint32 start_time = static_cast<Uint32>(tmp);
282                             Uint32 end_time = SDL_GetTicks();
283                             screen->SetLatency(end_time - start_time);
284                         }
285                         break;
286                     case urbi::DATA_STRING:
287                         std::cout << "String: " << *(value.stringValue) << std::endl;
288                         break;
289                     case urbi::DATA_BINARY:
290                         std::cout << "Binary: " << value.binary << std::endl;
291                         break;
292                     case urbi::DATA_LIST:
293                         std::cout << "List: " << value.list << std::endl;
294                         break;
295                     case urbi::DATA_OBJECT:
296                         std::cout << "Object: " << value.object << std::endl;
297                         break;
298                     case urbi::DATA_VOID:
299                         std::cout << "Void: " << value.storage << std::endl;
300                         break;
301                 }
302         }
303         return URBI_CONTINUE;
304     }
```

Listing 5: callbacks.cpp

A.6 callbackdata.hpp

```
1  #ifndef CALLBACKDATA_HPP
2  #define CALLBACKDATA_HPP
3
4  #include "urbi/fwd.hh"
5
6  namespace Aibo {
7
8  class Screen;
9  class Urbi;
10
11 // structure for passing data to callback functions
12 struct CallbackData {
13     CallbackData(Screen *s, Urbi *u, urbi::UClient *c);
14     Screen *screen;
15     Urbi *urbi;
16     urbi::UClient *client;
17     bool send_events;
18 };
19
20 } // namespace Aibo
21
22 #endif /* CALLBACKDATA_HPP */
```

Listing 6: callbackdata.hpp

A.7 callbackdata.cpp

```
1  #include "callbackdata.hpp"
2
3  #include "urbi/uclient.hh"
4
5  using namespace std;
6  using namespace urbi;
7  using namespace Aibo;
```

```
8
9  CallbackData::CallbackData(Screen *s, Urbi *u, UClient *c)
10     : screen(s), urbi(u), client(c),
11       send_events(false)
12 {}
```

Listing 7: callbackdata.cpp

A.8 urbistream.hpp

```
1  #ifndef URBI_STREAM_HPP
2  #define URBI_STREAM_HPP
3
4  #include "urbi/uabstractclient.hh"
5
6  #include <iostream>
7
8  namespace Aibo {
9
10 void StreamMessage(std::ostream& os, const urbi::UMessage& msg);
11 void StreamValue(std::ostream& ss, const urbi::UValue& value);
12 void StreamObject(std::ostream& os, const urbi::UObjectStruct& obj);
13
14 } // namespace Aibo
15
16 #endif /* URBI_STREAM_HPP */
```

Listing 8: urbistream.hpp

A.9 urbistream.cpp

```
1  #include "urbi_stream.hpp"
2
3  #include "urbi/uabstractclient.hh"
4  #include "urbi/uclient.hh"
5  #include "urbi/ubinary.hh"
6  #include "urbi/ublend-type.hh"
7  #include "urbi/ucallbacks.hh"
8  #include "urbi/uconversion.hh"
9  #include "urbi/uexternal.hh"
10 #include "urbi/uobject.hh"
11 #include "urbi/uproperty.hh"
12 #include "urbi/ustarter.hh"
13 #include "urbi/usystem.hh"
14 #include "urbi/utag.hh"
15 #include "urbi/uvalue.hh"
16 #include "urbi/uvar.hh"
17
18 #include <iostream>
19
20 using namespace std;
21 using namespace urbi;
22 using namespace Aibo;
23
24 void Aibo::StreamMessage(ostream& os, const UMessage& msg)
25 {
26     os << "[" << msg.timestamp << ":" << msg.tag << "]";
27     switch (msg.type) {
28         case urbi::MESSAGE_SYSTEM:
29             os << " *** " << msg.message;
30             break;
31         case urbi::MESSAGE_ERROR:
32             os << " !!! " << msg.message;
33             break;
34         case urbi::MESSAGE_DATA:
35             StreamValue(os, *(msg.value));
36             break;
37     }
38 }
39
```

```
40  void Aibo::StreamValue(ostream& os, const UValue& value)
41  {
42      switch (value.type) {
43          case DATA_DOUBLE:
44              os << "Double: " << value.val;
45              break;
46          case DATA_STRING:
47              os << "String: " << *(value.stringValue);
48              break;
49          case DATA_BINARY:
50              os << "Binary: " << value.binary;
51              break;
52          case DATA_LIST:
53              os << "List: " << value.list;
54              break;
55          case DATA_OBJECT:
56              os << "Object: " << value.object;
57              StreamObject(os, *(value.object));;
58              break;
59          case DATA_VOID:
60              os << "Void: " << value.storage;
61              break;
62      }
63  }
64
65  void Aibo::StreamObject(ostream& os, const UObjectStruct& cobj)
66  {
67      UObjectStruct& obj = const_cast<UObjectStruct&>(cobj);
68      for (int i = 0; i < obj.size(); ++i) {
69          const UNamedValue& nv = obj[i];
70          os << " Named value"  << "(" << i << ")" << ": " << nv.name;
71          StreamValue(os, *(nv.val));
72      }
73  }
```

Listing 9: urbistream.cpp

A.10 events.hpp

```
1   #ifndef EVENTS_HPP
2   #define EVENTS_HPP
3
4   namespace Aibo {
5
6   class CallbackData;
7
8   // Processes events from the OS.
9   class Events {
10  public:
11      Events(CallbackData *cbdata);
12      void Process();
13  private:
14      CallbackData *cbdata;
15  };
16
17  } // namespace Aibo
18
19  #endif /* EVENTS_HPP */
```

Listing 10: events.hpp

A.11 events.cpp

```
1   #include "events.hpp"
2   #include "screen.hpp"
3   #include "callbacks.hpp"
4   #include "callbackdata.hpp"
5   #include "urbi.hpp"
6
7   #include "SDL.h"
```

```
 8
 9  #include "urbi/uabstractclient.hh"
10  #include "urbi/uclient.hh"
11
12  using namespace std;
13  using namespace urbi;
14  using namespace Aibo;
15
16  Events::Events(CallbackData *cbdata)
17      : cbdata(cbdata)
18  {
19      Screen& screen = *(cbdata->screen);
20      screen.SetSending(cbdata->send_events);
21  }
22
23  void Events::Process()
24  {
25      UClient& client = *(cbdata->client);
26      Screen& screen = *(cbdata->screen);
27
28      bool done = false;
29      SDL_Event event;
30
31      while (!done) {
32          screen.SetSending(cbdata->send_events);
33          while(SDL_PollEvent(&event)) {  /* Loop until there are no events left on the
                queue */
34              switch(event.type) { /* Process the appropiate event type */
35                  case SDL_KEYDOWN:  /* Handle a KEYDOWN event */
36                      printf("Key pressed\n");
37                      switch (event.key.keysym.sym) {
38                          case SDLK_UP:
39                              screen.LineUp();
40                              break;
41                          case SDLK_DOWN:
42                              screen.LineDown();
43                              break;
44                          case SDLK_F1:
45                              client.send("emit handleft;");
46                              break;
47                          case SDLK_F2:
48                              client.send("emit handright;");
49                              break;
50                          case SDLK_F3:
51                              client.send("emit bodyleft;");
52                              break;
53                          case SDLK_F4:
54                              client.send("emit bodyright;");
55                              break;
56                          case SDLK_F5:
57                              client.send("emit up;");
58                              break;
59                          case SDLK_F6:
60                              client.send("emit down;");
61                              break;
62                          case SDLK_F7:
63                              client.send("emit idle;");
64                              break;
65                          case SDLK_F12:
66                              client.send("emit simulate;");
67                              break;
68                          case SDLK_TAB:
69                              cbdata->send_events = !cbdata->send_events;
70                              screen.SetSending(cbdata->send_events);
71                              break;
72                          case SDLK_ESCAPE:
73                              cbdata->urbi->ResendScripts();
74                              break;
75                          default:
76                              ;
77                      }
78                      break;
79                  case SDL_USEREVENT:
80                      switch (event.user.code) {
81                          case 0:
82                              client.send("latency << %d;", SDL_GetTicks());
83                              break;
```

```
84                      }
85                      break;
86                  case SDL_QUIT:
87                      done = true;
88                      break;
89                  default: /* Report an unhandled event */
90                      printf("Unhandled event.\n");
91              }
92          }
93          SDL_Delay(10);
94      }
95  }
```

Listing 11: events.cpp

A.12 screen.hpp

```
1   #ifndef SCREEN_H
2   #define SCREEN_H
3
4   #include "SDL.h"
5   #include "SDL_ttf.h"
6
7   #include <string>
8   #include <vector>
9
10  namespace Aibo {
11
12  // Handles updates to the screen.
13  class Screen {
14  public:
15      Screen(int width, int height, int bpp);
16      ~Screen();
17
18      void SetImage(void *data, int width, int height);
19      void PrintLine(const std::string& line);
20      void SetCPULoad(float load);
21      void SetFreeMemory(float freemem);
22      void SetPowerLevel(float level);
23      void SetLatency(int latency);
24      void SetGesture(const char *gesture);
25      void SetSending(bool sending);
26
27      void LineUp();
28      void LineDown();
29  private:
30      enum { FPSLine, CPULoadLine, FreeMemoryLine, PowerLevelLine, LatencyLine,
            GestureLine, SendingLine };
31
32      SDL_Surface *screen;
33      int screen_width;
34      int screen_height;
35      int image_width;
36      int image_height;
37
38      Uint32 lasttime;
39      Uint32 timepassed;
40      int fps;
41      int framecount;
42
43      void UpdateFps();
44      void PrintFps();
45
46      TTF_Font *font;
47
48      void PutSolid(int x, int y, const std::string& txt);
49
50      struct Line {
51          Line(const std::string& txt)
52              : text(txt), surface(NULL) {}
53          std::string text;
54          SDL_Surface *surface;
55      };
56      std::vector<Line> lines;
```

```
57      void PrintLines();
58      int start_line;
59  };
60
61  } // namespace Aibo
62
63  #endif /* SCREEN_H */
```

Listing 12: screen.hpp

A.13 screen.cpp

```
1   #include "screen.hpp"
2
3   #include "SDL.h"
4   #include "SDL_ttf.h"
5
6   #include <iostream>
7   #include <sstream>
8   #include <string>
9   #include <vector>
10  #include <cassert>
11
12  using namespace std;
13  using namespace Aibo;
14
15  static const SDL_Color white = { 0xff, 0xff, 0xff, 0x00 };
16  static const SDL_Color black = { 0x00, 0x00, 0x00, 0x00 };
17
18  Screen::Screen(int width, int height, int bpp)
19      : screen(NULL), screen_width(width), screen_height(height), image_width(0),
            image_height(0),
20        lasttime(0), timepassed(0), fps(0), framecount(0),
21        font(NULL), start_line(0)
22  {
23      // initialize video
24      SDL_Init(SDL_INIT_VIDEO | SDL_INIT_TIMER);
25      screen = SDL_SetVideoMode(width, height, bpp, SDL_SWSURFACE);
26
27      // initialize font
28      TTF_Init();
29      font = TTF_OpenFont("FreeSans.ttf", 12);
30      if (!font) {
31          cerr << "Error opening font: " << TTF_GetError() << endl;
32      }
33
34      // set key repeat
35      if (SDL_EnableKeyRepeat(SDL_DEFAULT_REPEAT_DELAY, SDL_DEFAULT_REPEAT_INTERVAL) ==
            -1) {
36          cerr << "Error setting key repeat" << endl;
37      }
38
39      lasttime = SDL_GetTicks();
40  }
41
42  Screen::~Screen()
43  {
44      TTF_CloseFont(font);
45      TTF_Quit();
46      SDL_Quit();
47  }
48
49  void Screen::SetImage(void *data, int width, int height)
50  {
51      if (image_width != width || image_height != height) {
52          image_width = width;
53          image_height = height;
54          SDL_FillRect(screen, NULL, 0);
55          SDL_UpdateRect(screen, 0, 0, screen->w, screen->h);
56          PrintLines();
57          SetGesture("none");
58      }
59
60      UpdateFps();
```

```
61
62      SDL_Surface *image_surface = SDL_CreateRGBSurfaceFrom(data, width, height, 24, 3*
            width+0, 0xff0000, 0x00ff00, 0x0000ff, 0x000000);
63
64      SDL_Rect src = { 0, 0, width, height };
65      SDL_Rect dest = { 0, 0, width, height };
66      SDL_BlitSurface(image_surface, &src, screen, &dest);
67      SDL_UpdateRect(screen, dest.x, dest.y, dest.w, dest.h);
68
69      SDL_FreeSurface(image_surface);
70
71      PrintFps();
72  }
73
74  void Screen::PrintLine(const string& line)
75  {
76      cout << line << endl;
77      lines.push_back(Line(line));
78
79      PrintLines();
80  }
81
82  void Screen::SetCPULoad(float load)
83  {
84      stringstream ss;
85      ss << "CPU load: " << load*100 << " %    ";
86      PutSolid(image_width, CPULoadLine*TTF_FontLineSkip(font), ss.str());
87  }
88
89  void Screen::SetFreeMemory(float freemem)
90  {
91      stringstream ss;
92      ss << "Free memory: " << (freemem / 1024) / 1024 << " Mbytes    ";
93      PutSolid(image_width, FreeMemoryLine*TTF_FontLineSkip(font), ss.str());
94  }
95
96  void Screen::SetPowerLevel(float level)
97  {
98      stringstream ss;
99      ss << "Power level: " << level*100 << " %    ";
100     PutSolid(image_width, PowerLevelLine*TTF_FontLineSkip(font), ss.str());
101 }
102
103 void Screen::SetLatency(int latency)
104 {
105     stringstream ss;
106     ss << "Latency: " << latency << " ms        ";
107     PutSolid(image_width, LatencyLine*TTF_FontLineSkip(font), ss.str());
108 }
109
110 void Screen::SetGesture(const char *gesture)
111 {
112     stringstream ss;
113     ss << "Gesture: " << gesture << "                ";
114     PutSolid(image_width, GestureLine*TTF_FontLineSkip(font), ss.str());
115 }
116
117 void Screen::SetSending(bool sending)
118 {
119     string str;
120     if (sending) {
121         str = "Sending gesture events.       ";
122     } else {
123         str = "Not sending gesture events.   ";
124     }
125     PutSolid(image_width, SendingLine*TTF_FontLineSkip(font), str);
126 }
127
128 void Screen::LineUp()
129 {
130     --start_line;
131     if (start_line < 0)
132         start_line = 0;
133     PrintLines();
134 }
135
136 void Screen::LineDown()
```

```
137 {
138     ++start_line;
139     if (start_line >= (int)lines.size())
140         start_line = lines.size() - 1;
141     PrintLines();
142 }
143
144 void Screen::UpdateFps()
145 {
146     ++framecount;
147     timepassed = SDL_GetTicks() - lasttime;
148     if (timepassed > 1000) {
149         fps = (framecount * 1000) / timepassed;
150         lasttime = SDL_GetTicks();
151         framecount = 0;
152     }
153 }
154
155 void Screen::PrintFps()
156 {
157     stringstream ss;
158     ss << "Fps: " << fps << "    ";
159     PutSolid(image_width, FPSLine*TTF_FontLineSkip(font), ss.str());
160 }
161
162 void Screen::PutSolid(int x, int y, const std::string& txt)
163 {
164     SDL_Surface *text_surface = TTF_RenderText_Solid(font, txt.c_str(), white);
165     if (!text_surface) {
166         cerr << "Error rendering text: " << TTF_GetError() << endl;
167     }
168     SDL_Rect src = { 0, 0, text_surface->w, text_surface->h };
169     SDL_Rect dest = { x, y, text_surface->w, text_surface->h };
170     SDL_FillRect(screen, &dest, 0);
171     SDL_BlitSurface(text_surface, &src, screen, &dest);
172     SDL_UpdateRect(screen, dest.x, dest.y, dest.w, dest.h);
173     SDL_FreeSurface(text_surface);
174 }
175
176 void Screen::PrintLines()
177 {
178     SDL_Rect text_area = { 0, image_height, screen_width, screen_height };
179     SDL_Rect dest = { 0, image_height, 0, 0 };
180     SDL_FillRect(screen, &text_area, 0);
181     for (unsigned int i = start_line; i < lines.size(); ++i) {
182         if (lines[i].surface == 0) {
183             lines[i].surface = TTF_RenderText_Blended(font, lines[i].text.c_str(), white
                    );
184         }
185         SDL_Rect src = { 0, 0, lines[i].surface->w, lines[i].surface->h };
186         dest.x = 0;
187         dest.y += + TTF_FontLineSkip(font);
188         dest.w = lines[i].surface->w;
189         dest.h = lines[i].surface->h;
190         SDL_BlitSurface(lines[i].surface, &src, screen, &dest);
191     }
192     SDL_UpdateRect(screen, text_area.x, text_area.y, text_area.w, text_area.h);
193 }
```

Listing 13: screen.cpp

A.14 timer.hpp

```
1  #ifndef TIMER_H
2  #define TIMER_H
3
4  #include "SDL.h"
5
6  namespace Aibo {
7
8  class Timer {
9  public:
10     explicit Timer(Uint32 delay, int code = 0);
11     ~Timer();
```

```
12  private:
13      static Uint32 callbackfunc(Uint32 interval, void *param);
14      SDL_TimerID id;
15      int code;
16  };
17
18  } // namespace Aibo
19
20  #endif /* TIMER_H */
```

Listing 14: timer.hpp

A.15 timer.cpp

```
1   #include "timer.hpp"
2
3   #include "SDL.h"
4
5   #include <iostream>
6
7   using namespace std;
8   using namespace Aibo;
9
10  Timer::Timer(Uint32 delay, int code)
11      : id(0), code(code)
12  {
13      id = SDL_AddTimer(delay, callbackfunc, this);
14  }
15
16  Timer::~Timer()
17  {
18      SDL_RemoveTimer(id);
19  }
20
21  Uint32 Timer::callbackfunc(Uint32 interval, void *param)
22  {
23      Timer *timer = static_cast<Timer *>(param);
24
25      SDL_UserEvent userevent;
26      userevent.type = SDL_USEREVENT;
27      userevent.code = timer->code;
28      userevent.data1 = timer;
29      userevent.data2 = NULL;
30
31      SDL_Event event;
32      event.type = SDL_USEREVENT;
33      event.user = userevent;
34
35      SDL_PushEvent(&event);
36      return interval;
37  }
```

Listing 15: timer.cpp

B Urbi program

B.1 aibo.cpp

```
1   # list of scripts to send in order
2
3   # library scripts
4   eyes.u
5   bark.u
6   paws.u
7   tail.u
8   ears.u
9   support.u
10
```

```
11  # main action scripts
12  hands.u
13  body.u
14  up.u
15  down.u
16  idle.u
17
18  # setup and run scripts
19  camera.u
20  loops.u
21  setup.u
22  events.u
23
24  # simulate event
25  simulate.u
```

Listing 16: scripts.lst

B.2 eyes.u

```
1   log << echo("Defining Eyes object...");
2
3   if (isdef(eyes)) {
4       delete eyes;
5   };
6
7   if (isdef(Eyes)) {
8       undef Eyes;
9       undef Eyes.Open;
10      undef Eyes.Close;
11      undef Eyes.Blink;
12  };
13
14  class Eyes {
15      function init();
16      function Open();
17      function Close();
18      function Blink(times);
19  };
20
21  function Eyes.init()
22  {
23      global.ledMode = 0;
24      ledF.val = 0;
25  };
26
27  function Eyes.Open()
28  {
29      ledF14.val = 1 &
30      ledF13.val = 0 &
31      ledF7.val = 1 &
32      ledF8.val = 1;
33  };
34
35  function Eyes.Close()
36  {
37      ledF14.val = 1 &
38      ledF13.val = 1 &
39      ledF7.val = 0 &
40      ledF8.val = 0;
41  };
42
43  function Eyes.Blink(times)
44  {
45      var count = times;
46      while (count > 0) {
47          self.Close() | wait(300ms) | self.Open() | wait(200ms) |
48          count = count - 1;
49      };
50  };
```

Listing 17: eyes.u

B.3 bark.u

```
1  log << echo("Defining Bark object...");
2
3  if (isdef(bark)) {
4      delete bark;
5  };
6
7  if (isdef(Bark)) {
8      undef Bark;
9      undef Bark.init;
10     undef Bark.Once;
11     undef Bark.Double;
12     undef Bark.Hard;
13 };
14
15 class Bark {
16     function init();
17     function Once();
18     function Double();
19     function Hard();
20 };
21
22 function Bark.init()
23 {
24 };
25
26 function Bark.Once()
27 {
28     mouth.val'n = 1 |
29     {
30         speaker.play("bark.wav") &
31         mouth.val'n = 0.25 smooth:200ms;
32     } |
33     mouth.val'n = 1 smooth:100ms;
34 };
35
36 function Bark.Double()
37 {
38     self.Once() | self.Once();
39 };
40
41 function Bark.Hard()
42 {
43     mouth.val'n = 1 |
44     {
45         speaker.play("bark2.wav") &
46         mouth.val'n = 0 smooth:200ms |
47         wait(100ms);
48     } |
49     mouth.val'n = 1 smooth:100ms;
50 };
```

Listing 18: bark.u

B.4 paws.u

```
1  log << echo("Defining Paws object...");
2
3  if (isdef(paws)) {
4      delete paws;
5  };
6
7  if (isdef(Paws)) {
8      undef Paws;
9      undef Paws.RaiseLeftFront;
10     undef Paws.RaiseRightFront;
11     undef Paws.Lower;
12     undef Paws.LowerLeftFront;
13     undef Paws.LowerRightFront;
14     undef Paws.SafeOld;
15 };
16
```

```
17  class Paws {
18      function init();
19      function RaiseLeftFront();
20      function RaiseRightFront();
21      function Lower();
22      function LowerLeftFront();
23      function LowerRightFront();
24      function SafeOld();
25      var left_raised;
26      var right_raised;
27      var old_legLF1_val;
28      var old_legLF2_val;
29      var old_legLF3_val;
30      var old_legRF1_val;
31      var old_legRF2_val;
32      var old_legRF3_val;
33  };
34
35  function Paws.init()
36  {
37      self.left_raised = false;
38      self.right_raised = false;
39  };
40
41  function Paws.RaiseLeftFront()
42  {
43      if (self.right_raised) {
44          self.LowerRightFront() |
45      } |
46      if (!self.left_raised) {
47          self.left_raised = true |
48          self.SafeOld() |
49
50          // raise paw
51          {
52              legLF3.val'n = 0.6 smooth:1s &
53              {
54                  wait(500ms) |
55                  {
56                      legLF1.val'n = 0.75 smooth:1s &
57                      legLF2.val'n = 0.5 smooth:1s
58                  }
59              } |
60          } |
61          { // stretch paw
62              legLF1.val'n = 0.4 smooth:1s &
63              legLF2.val'n = 1 smooth:1s &
64              {
65                  wait(500ms) |
66                  legLF3.val = 10 smooth:1s;
67              }
68          }
69      };
70  };
71
72  function Paws.RaiseRightFront()
73  {
74      if (self.left_raised) {
75          self.LowerLeftFront() |
76      } |
77      if (!self.right_raised) {
78          self.right_raised = true |
79          self.SafeOld() |
80
81          // raise paw
82          {
83              legRF3.val'n = 0.6 smooth:1s &
84              {
85                  wait(500ms) |
86                  {
87                      legRF1.val'n = 0.75 smooth:1s &
88                      legRF2.val'n = 0.5 smooth:1s |
89                  } |
90              } |
91          } |
92          { // stretch paw
93              legRF1.val'n = 0.4 smooth:1s &
```

```
94              legRF2.val'n = 1 smooth:1s &
95              {
96                  wait(500ms) |
97                  legRF3.val = 10 smooth:1s;
98              }
99          }
100     };
101 };
102
103 function Paws.Lower()
104 {
105     if (self.left_raised) {
106         self.LowerLeftFront();
107     };
108     if (self.right_raised) {
109         self.LowerRightFront();
110     };
111 };
112
113 function Paws.LowerLeftFront()
114 {
115     {
116         legLF2.val = self.old_legLF2_val smooth:1s &
117         legLF3.val'n = 0.6 smooth:500ms
118     } |
119     legLF1.val = self.old_legLF1_val smooth:1s |
120     legLF3.val = self.old_legLF3_val smooth:1s |
121     self.left_raised = false;
122 };
123
124 function Paws.LowerRightFront()
125 {
126     {
127         legRF2.val = self.old_legRF2_val smooth:1s &
128         legRF3.val'n = 0.6 smooth:500ms
129     } |
130     legRF1.val = self.old_legRF1_val smooth:1s |
131     legRF3.val = self.old_legRF3_val smooth:1s |
132     self.right_raised = false;
133 };
134
135 function Paws.SafeOld()
136 {
137     self.old_legLF1_val = legLF1.val;
138     self.old_legLF2_val = legLF2.val;
139     self.old_legLF3_val = legLF3.val;
140     self.old_legRF1_val = legRF1.val;
141     self.old_legRF2_val = legRF2.val;
142     self.old_legRF3_val = legRF3.val;
143 };
```

Listing 19: paws.u

B.5 tail.u

```
1  log << echo("Defining Tail object...");
2
3  if (isdef(_tail)) {
4      delete _tail;
5  };
6
7  if (isdef(Tail)) {
8      undef Tail;
9      undef Tail.init;
10     undef Tail.Raise;
11     undef Tail.Lower;
12     undef Tail.Wag;
13     undef Tail.StopWag;
14 };
15
16 class Tail {
17     function init();
18     function Raise();
19     function Lower();
```

```
20      function Wag(speed);
21      function StopWag();
22      var slow;
23      var medium;
24      var fast;
25  };
26
27  function Tail.init()
28  {
29      self.slow = 5s;
30      self.medium = 1s;
31      self.fast = 500ms;
32  };
33
34  function Tail.Raise()
35  {
36      tailPan.val = 0 smooth:50ms &
37      tailTilt.val = 2 smooth:200ms;
38  };
39
40  function Tail.Lower()
41  {
42      tailPan.val = 0 smooth:200ms &
43      tailTilt.val = 63 smooth:500ms;
44  };
45
46  function Tail.Wag(speed)
47  {
48      tailwagging: tailPan.val = 0 sin:speed ampli:40,
49  };
50
51  function Tail.StopWag()
52  {
53      stop tailwagging;
54  };
```

Listing 20: tail.u

B.6 ears.u

```
1   log << echo("Defining Ears object...");
2
3   if (isdef(_ears)) {
4       delete _ears;
5   };
6
7   if (isdef(Ears)) {
8       undef Ears;
9       undef Ears.init;
10      undef Ears.Raise;
11      undef Ears.Lower;
12  };
13
14  class Ears {
15      function init();
16      function Raise();
17      function Lower();
18  };
19
20  function Ears.init()
21  {
22  };
23
24  function Ears.Raise()
25  {
26      earL.val = 1 & earR.val = 1;
27  };
28
29  function Ears.Lower()
30  {
31      earL.val = 0 & earR.val = 0;
32  };
```

Listing 21: ears.u

B.7 support.u

```
// puts string s in qoutes
function quote(s)
{
    return "\"" + s + "\"";
};

// create support objects
eyes = new Eyes;
bark = new Bark;
paws = new Paws;
_tail = new Tail;
_ears = new Ears;

// allow background tailwagging
event wagtail;
event stopwagtail;

at (wagtail(speed)) {
    _tail.Wag(speed),
},

at (stopwagtail()) {
    _tail.StopWag() |
    _tail.Raise();
},

log << echo("Support objects created.");
```

Listing 22: support.u

B.8 hands.u

```
if (isdef(action.hands_left)) {
    undef action.do_hands_left;
    undef action.do_hands_right;
    undef action.undo_hands_left;
    undef action.undo_hands_right;
};

function action.do_hands_left(posture, gesture)
{
    if (posture != "sitting") {
        robot.SitDown()
    } |
    paws.RaiseLeftFront() |
    headTilt.val = 35 smooth:400ms;
};

function action.undo_hands_left(posture, gesture)
{
    paws.LowerLeftFront();
};

function action.do_hands_right(posture, gesture)
{
    if (posture != "sitting") {
        robot.SitDown()
    } |
    paws.RaiseRightFront() |
    headTilt.val = 35 smooth:400ms;
};

function action.undo_hands_right(posture, gesture)
{
    paws.LowerRightFront();
};
```

Listing 23: hands.u

B.9 body.u

```
if (isdef(action.do_body_right)) {
    undef action.do_body_right;
    undef action.undo_body_right;
    undef action.do_body_left;
    undef action.undo_body_left;
};

function action.do_body_right(posture, gesture)
{
    if (posture != "standing") {
        robot.StandUp()
    };
    robot.sturn(1) & headPan.val = 50 smooth: 5s |
    neck.val = 0 smooth:400ms & headTilt.val = 20 smooth:400ms;
};

function action.undo_body_right(posture, gesture)
{
    robot.sturn(-1) & headPan.val = 0 smooth: 5s;
};

function action.do_body_left(posture, gesture)
{
    if (posture != "standing") {
        robot.StandUp()
    };
    robot.sturn(-1) & headPan.val = -50 smooth: 5s |
    neck.val = 0 smooth:400ms & headTilt.val = 20 smooth:400ms;
};

function action.undo_body_left(posture, gesture)
{
    robot.sturn(1) & headPan.val = 0 smooth: 5s;
};
```

Listing 24: body.u

B.10 up.u

```
if (isdef(action.do_up)) {
    undef action.do_up;
    undef action.undo_up;
};

function action.do_up(posture, gesture)
{
    if (posture != "standing") {
        robot.StandUp()
    } |
    { _ears.Raise() & _tail.Raise() & eyes.Blink(3) } |
    { neck.val = 0 smooth:400ms & headTilt.val = 20 smooth:400ms } |
    emit wagtail(_tail.medium);
};

function action.undo_up(posture, gesture)
{
    emit stopwagtail &
    _ears.Lower() &
    eyes.Open();
};
```

Listing 25: up.u

B.11 down.u

```
if (isdef(action.do_down)) {
    undef action.do_down;
    undef action.undo_down;
```

```
4   };
5
6   function action.do_down(posture, gesture)
7   {
8       if (posture != "lying") {
9           robot.LayDown()
10      } &
11      _ears.Lower() &
12      _tail.Lower() |
13      headTilt.val = 30 smooth:400ms;
14  };
15
16  function action.undo_down(posture, gesture)
17  {
18      _tail.Raise();
19  };
```

Listing 26: down.u

B.12 idle.u

```
1   if (isdef(action.do_idle)) {
2       undef action.do_idle;
3   };
4
5   function action.do_idle(posture, gesture)
6   {
7       if (posture != "lying") {
8           robot.LayDown()
9       } &
10      _tail.Lower() &
11      _ears.Lower() &
12      eyes.Blink(4);
13      headTilt.val = 30 smooth:400ms;
14  };
15
16  function action.undo_idle(posture, gesture)
17  {
18      _tail.Raise() &
19      eyes.Open();
20  };
```

Listing 27: idle.u

B.13 camera.u

```
1   log << echo("Setting up camera...");
2
3   // setup camera
4   setup: camera.shutter = 1;       // The camera shutter speed: 1=SLOW (default), 2=MID, 3=
        FAST.
5   setup: camera.wb = 1;            // The camera white balance: 1=INDOOR (default), 2=
        OUTDOOR, 3=FLUO.
6   setup: camera.gain = 3;          // The camera gain: 1=LOW, 2=MID, 3=HIGH (default).
7   setup: camera.format = 1;        // The camera image format: 0=YCbCr 1=jpeg (default).
8   setup: camera.jpegfactor = 80;   // The jpeg compression factor (0 to 100). Default=80.
9   setup: camera.resolution = 0;    // The image resolution: 0:208x160 (default) 1:104x80
        2:52x40.
10  setup: camera.reconstruct = 0;   // Reconstruction of the high resolution image(slow): 0:
        no (default) 1:yes.
11
12  // setup camera message loop
13  camera_loop:
14      every(50ms) image << camera.val,
15
16  log << echo("Camera loop running.");
```

Listing 28: camera.u

B.14 loops.u

```
1  log << echo("Starting various loops...");
2
3  // setup message loops
4  load_loop:
5      every(100ms) {
6          load << cpuload();
7      },
8
9  freemem_loop:
10     every(500ms) {
11         memory << freemem();
12     },
13
14 power_loop:
15     every(1000ms) {
16         powerlevel << power();
17     },
18
19 // send first latency message
20 //setup: at (latency_event()) latency << ping,
21 //latency_loop: every(1s) emit latency_event,
22
23 log << echo("Loops running.");
```

Listing 29: loops.u

B.15 setup.u

```
1  log << echo("Starting setup...");
2
3  // turn on motors
4  setup: motor on;
5
6  // let aibo signal that script is running
7  setup:
8      {
9          neck.val'n = 1 |
10         neck.val'n = 0 time: 1000ms |
11         neck.val'n = 1 time: 500ms;
12     };
13
14 // put aibo in known starting state
15 setup:
16     {
17         robot.initial() |
18         {
19             eyes.Open() &
20             _ears.Lower() &
21             _tail.Raise() &
22             headTilt.val = 25 smooth:400ms;
23         };
24     };
25
26 log << echo("Motors on and initialized.");
```

Listing 30: setup.u

B.16 events.u

```
1  log << echo("Starting events...");
2
3  // register time of last event for idle event
4  var time_of_last_event = time();
5
6  // busy is true when aibo is moving
7  var busy = false;
8  busy->blend = queue;
9
```

```
10  whenever (busy) {
11      modeR.val = 1 & modeG.val = 0 & modeB.val = 0
12  } else {
13      modeR.val = 0 & modeG.val = 1 & modeB.val = 0
14  };
15
16  // the state aibo is now in
17  var state_posture = "initial";
18  var state_gesture = "none";
19
20  // action to execute, to undo the current gesture state
21  var previous_undo_action = "";
22
23  if (isdef(register_event)) {
24      undef register_event;
25      undef ignore_event;
26      undef run_event;
27      undef freeze_events;
28      undef unfreeze_events;
29  };
30
31  function register_event(s)
32  {
33      gesture << echo(s) &
34      time_of_last_event = time() &
35      bark.Double();
36  };
37
38  function ignore_event(s)
39  {
40      gesture << echo(s + " ignored") &
41      bark.Hard();
42  };
43
44  function freeze_events()
45  {
46      freeze left_hand_event &
47      freeze right_hand_event &
48      freeze body_left_event &
49      freeze body_right_event &
50      freeze up_event &
51      freeze down_event &
52      freeze idle_event;
53  };
54
55  function unfreeze_events()
56  {
57      unfreeze left_hand_event &
58      unfreeze right_hand_event &
59      unfreeze body_left_event &
60      unfreeze body_right_event &
61      unfreeze up_event &
62      unfreeze down_event &
63      unfreeze idle_event;
64  };
65
66  function run_event(name, posture, gesture, doaction, undoaction)
67  {
68      register_event(name) &
69      if (busy || (state_posture == posture && state_gesture == gesture) ) {
70          ignore_event(name);
71      } else {
72          busy = true |
73          if (previous_undo_action != "") {
74              var undocmd = previous_undo_action + "(\"" + state_posture + "\", \"" +
                    state_gesture + "\");" |
75              log << echo("execing: " + undocmd) |
76              exec(undocmd) |
77          } |
78          var docmd = doaction + "(\"" + state_posture + "\", \"" + state_gesture + "\");"
                |
79          log << echo("execing: " + docmd) |
80          exec(docmd) |
81          state_posture = posture |
82          state_gesture = gesture |
83          previous_undo_action = undoaction |
84          busy = false;
```

```
85      }
86  };
87
88  log << echo("Freezing events...");
89  freeze_events();
90
91  // on left hand: let aibo sit and raise front left paw
92  log << echo("Setting up left hand event...");
93  left_hand_event:
94      at (handleft()) {
95          run_event("hand left", "sitting", "lfpaw_up", "action.do_hands_left", "action.
                undo_hands_left");
96      },
97
98  // on right hand: let aibo sit and raise front right paw
99  log << echo("Setting up right hand event...");
100 right_hand_event:
101     at (handright()) {
102         run_event("hand right", "sitting", "rfpaw_up", "action.do_hands_right", "action.
                undo_hands_right");
103     },
104
105 // on left body turn: let aibo stand and turn left while keeping eyes on same spot
106 log << echo("Setting up body left event...");
107 body_left_event:
108     at (bodyleft()) {
109         run_event("body left", "standing", "look_left", "action.do_body_left", "action.
                undo_body_left");
110     },
111
112 // on right body turn: let aibo stand and turn right while keeping eyes on same spot
113 log << echo("Setting up body right event...");
114 body_right_event:
115     at (bodyright()) {
116         run_event("body right", "standing", "look_right", "action.do_body_right", "
                action.undo_body_right");
117     },
118
119 // on up: let aibo stand and raise ears and tail, let tail wag
120 log << echo("Setting up up event...");
121 up_event:
122     at (up()) {
123         run_event("up", "standing", "raised_ears_tail", "action.do_up", "action.undo_up"
                );
124     },
125
126 // on down: let aibo lie down with ears and tail down
127 log << echo("Setting up down event...");
128 down_event:
129     at (down()) {
130         run_event("down", "lying", "lowered_ears_tail", "action.do_down", "action.
                undo_down");
131     },
132
133 // on idle: put aibo in a lying position
134 log << echo("Setting up idle event...");
135 idle_event:
136     at (idle()) {
137         run_event("idle", "lying", "none", "action.do_idle", "action.undo_idle");
138     },
139
140 at (time() - time_of_last_event > 50000) {
141     emit idle;
142 },
143
144 log << echo("Unfreezing events...");
145 unfreeze_events();
146
147 log << echo("Events setup.");
```

Listing 31: events.u

B.17 simulate.u

```
1   simulate_event:
```

```
2  at (simulate()) {
3      log << echo("Starting simulation...");
4
5      wait(1s);
6
7      emit handleft;
8      wait(10s);
9
10     emit handright;
11     wait(10s);
12
13     emit bodyleft;
14     wait(15s);
15
16     emit bodyright;
17     wait(15s);
18
19     emit up;
20     wait(15s);
21
22     emit down;
23     wait(10s);
24
25     log << echo("Simulation ended.");
26 },
```

Listing 32: simulate.u

C Gesture library

C.1 transform.h

```
1  #ifndef TRANSFORM_H
2  #define TRANSFORM_H
3
4  #ifdef __cplusplus
5  namespace Gestures {
6  extern "C" {
7  #endif
8
9  typedef enum { NOERROR, ERROR } HRESULT;
10 typedef unsigned char BYTE;
11
12 typedef enum {
13     gesture_none,
14     gesture_body_left, gesture_body_right,
15     gesture_hand_left, gesture_hand_right,
16     gesture_up, gesture_down
17 } gestures_t;
18 extern const char *gestures_str[];
19 typedef void (*gesture_callback)(gestures_t gesture, void *data);
20
21 HRESULT Transform(BYTE *data, int width, int height, long len, gesture_callback cb, void
       *callback_data);
22 const char *GestureString(gestures_t gesture);
23 void PrintGesture(gestures_t gesture, void *data);
24
25 #ifdef __cplusplus
26 } // extern "c"
27 } // namespace Gestures
28 #endif
29
30 #endif /* TRANSFORM_H */
```

Listing 33: transform.h