Khepera Project Dresden

HTW-Dresden

Prof. Heino Iwe

Dominique Despang, Holger Ertel, Marko Haßler
René Liebscher, Oliver Michel, Falk Mölle
Daniel Popp, Konrad Rosenbaum, Tobias Schiebeck

April 25, 2000

The use of general descriptive names, trade names, trademarks, etc., in this publication, even if the former are not especially identified, is not to be taken as a sign that such names, as understood by the Trade Marks and Merchandise Marks Act, may accordingly be used freely by anyone.

Contents

Chapter 1

Introduction

This project was running as a 4th year project in computer science studies at the University of Applied Sciences in Dresden (*Hochschule für Technik und Wirtschaft Dresden (FH)* [1], Germany). The objective of this project, led by Prof. Heino Iwe [2], was to improve the robotics facillities in this faculty.

1.1 The Project

The aim of the Project was to implement a robot-simulator (EasyBot), a tool based on LightVision3D that should be able to simulate all kinds of robots (existing and imaginable). Other tasks were to develop an interface that can be used with a real Khepera robot as well as with the simulator, and several control units for the Khepera robot and simulator.

1.2 LightVision3D (LV3D)

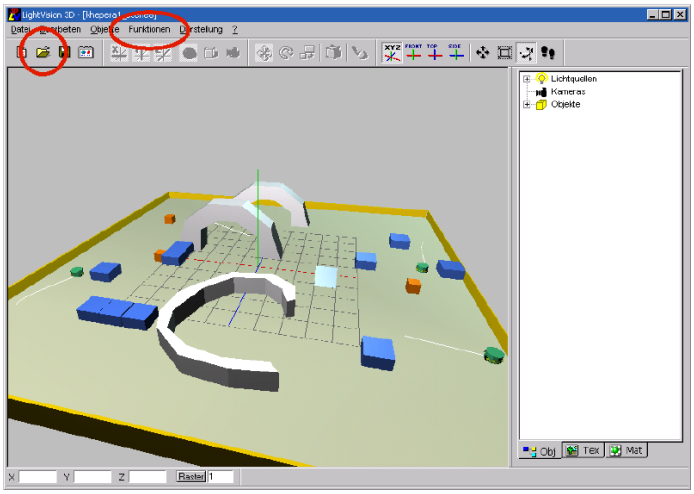
LightVision3D [3] is a 3D-viewer and -modeller, designed for beginners and advanced users like 3D-designers and -engineers, running in a Windows9x or WindowsNT environment. To get it working it is necessary to have the Internet Explorer (v.4.0 or higher) installed.

Features:

- objects:
  - mathematical objects (sphere, plane)
  - mesh-objects
    - described by vertexes and triangles
    - deformable using a magnet tool
  - group-objects
- lights (pointlight, spotlight)
- cameras
- materials, textures
- drag& drop to assign textures or materials to objects

LightVision3D is easy to use. Help can be obtained using the right mouse button. For further information (in German) about LightVision3D select “Hilfe/Hilfe” from the menu.

1.3 How to run the robots within LV3D ?



1. Installation
   Unzip the lv3d_easybot.zip to a directory on your harddisc.

2. Start LightVision3D
   Start LightVision3D with a doubleclick on LV3D.exe

3. Load a scene
   Load a robot-scene out of the “roboter”-directory (e.g. khepera1_scene8.lvs)

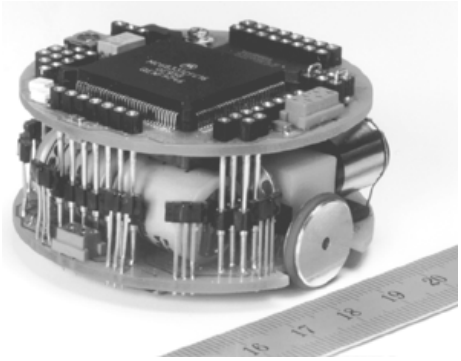
4. Start Easybot
   Call Easybot from the “Funktionen/Robotersimulation/Easybot”-menu. Easybot presents all the available robots in a list.

5. Select the navigation module
   For each robot you want to use, load a control-DLL. This can be done by a doubleclick on the name of the robot in the list.

   If DLLs are not visible in the filedialog, change the option of the Windows-Explorer that hides special files. This can be done by calling the Explorer, choosing “View/options”, and selecting “show all files”. Now restart LightVision3D and start over at point 3.

6. Danger! robots walking!
   Now, having assigned some DLLs to the robots, you can press the play-button “>” to see the robots in action.

1.4 The Khepera Robot



Khepera has originally been designed as a research and teaching tool in the framework of a Swiss Research Priority Program. It was developed at “Ecole polytechnique fédérale de Lausanne” in Swizerland. There are two modes to use this robot. You can either control it via a serial link, or download a compiled program, to let the robot work autonomously.

1.5 Contact the HTW-Khepera-Team

Prof. Heino Iwe
www.htw-dresden.de/ iwe/
iwe@informatik.htw-dresden.de

HTW-Dresden
Friedrich-List-Platz 1
D - 01069 Dresden
FB Informatik

Tel.: +49 351 462 0
Fax.: +49 351 462.21.85

EasyBot - Team:

www.htw-dresden.de/ iwe/easybot/easybot.html

Chapter 2

EasyBot - the Simulator

2.1 Introduction

EasyBot is a robot-simulator. It can be used for scientific research in the field of robotics, to develop algorithms for robot navigation.

EasyBot is an extension for LightVision3D to manipulate robot-objects. Robot-objects contain sensors, built of triangles, named "sensor…". All group-objects, containing such sensor-objects, are recognized as robots, which can be controlled by a DLL. The control must be inherited from the interface described below.

2.2 The User Interface

The numbered regions in figure 2.1 contain controls to manage following aspects of simulation:

1. List of available robots in the scene, that can be controlled. With a double-click you can open a file-dialog to choose a DLL.
2. Some options:

   "robot camera" Switches the 3D-output of LV3D to a robot related view. Within the combo-box you can choose the direction of the robot camera.

   "follow landscape" Lets the robots follow the landscape while moving around. This enables the robots to operate in non-flat environments. However this does not work with steps and steep rises.

   "show track" Shows the track of the robot for the last simulated steps.

   "save positions" Saves some data of each simulation-step for later analysis.

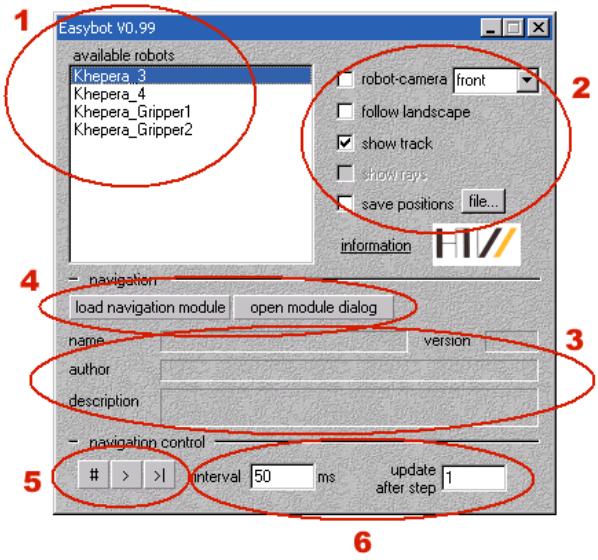


Figure 2.1: EasyBot User Interface

3. Displays (about-) information of the control-DLL.

4. With the left button you can load a control-DLL for the selected robot. The other one opens a configuration dialog for the current robot, if implemented within the control-DLL.

5. Simulation control:

#	stops the simulation.
>	starts the simulation.
>|	executes one simulation step.

6. The “interval” is the time-base of the simulation. This is the minimum time between the starts of two simulation steps. LV3D ignores this value, if it needs more time to update the display. The “update after step”-value defines the number of steps between two display updates.

2.3 Example for Writing a Control-DLL for EasyBot

This chapter wants to explain, how to write a DLL, which controls a robot in EasyBot. The robot, we want control, is a Khepera with a K213-extension. The interface explained here, is the basic interface of EasyBot.

EasyBot provides a general robot interface. Basing on this interface, it is possible to write interfaces for real robots. Later in this document we describe the interface for the Khepera robot (see 3.1).

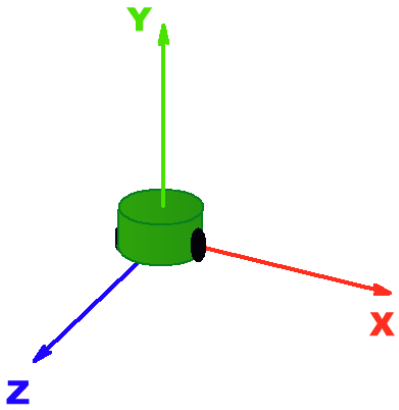
2.3.1 The Development Environment

The development environment is Microsoft Visual Studio 5.0/6.0. We suppose, you can also use other development environments, that can produce DLL's.

2.3.2 The Khepera Control Interface

The model of the robot contains 8+64 ray-sensors, which represent the sensors of the Khepera and the K213-extension. The first 8 sensors are the IR-sensors of the Khepera. For each pixel of the K213-one-line-camera exists one ray-sensor (that are the other 64 sensors). The sensors are organized in an 72 (8+64) element-array of `easybot_ray_info`-structures.

The reaction on the sensor-data is implemented within the `Navigate()`-function of your DLL. Possible reactions are operations like `move` along or `rotate` around the X,Y and Z-axis. Usually the Khepera only rotates around the Y-axis, and moves along the Z-axis.



2.3.3 The Interface

The example below performs a line tracking algorithm as described later (see 4.2).

Your controll-DLL must inherit the interface `DLL_Interface_Easybot`. All abstract functions have to be implemented.

```
class DLL_Interface_Easybot
{
public:
   virtual char*   __cdecl GetName()=0;
   virtual char*   __cdecl GetVersion()=0;
   virtual char*   __cdecl GetAuthor()=0;
   virtual char*   __cdecl GetDescription()=0;

   virtual void    __cdecl SetEasybotInfo
                           (Easybot_Interfaces *p)=0;
   virtual void    __cdecl SetRayInfo(easybot_ray_info *p,
                              long highest_index)=0;
   virtual void    __cdecl SetObjectInfo
                           (easybot_object_info *p)=0;

   virtual void    __cdecl SetMessageFrom(char *pfrom_robot,
                                 char *pmessage)=0;
   virtual int     __cdecl GetMessageFor(char **ppto_robot,
                                char **ppmessage)=0;

   virtual void    __cdecl Navigate()=0;

   virtual float   __cdecl GetMoveX()=0;
   virtual float   __cdecl GetMoveY()=0;
   virtual float   __cdecl GetMoveZ()=0;
   virtual float   __cdecl GetRotationX()=0;
   virtual float   __cdecl GetRotationY()=0;
   virtual float   __cdecl GetRotationZ()=0;

   virtual void    __cdecl OpenDialog()=0;
   virtual void    __cdecl SetStartStop(int onoff)=0;
   virtual void    __cdecl Destroy()=0;
};
#endif
```

The call-convention `__cdecl` is used to be compatible with Borland Delphi.

The methods `GetName()`, `GetVersion()`, `GetAuthor()`, `GetDescription()`, `SetEasybotInfo()`, `SetRayInfo()` and `SetObjectInfo()` are executed once, when the object is constructed. `Destroy()` is called, when the user closes EasyBot.

Function	Description
GetName	returns the name of your DLL
GetVersion	returns a Version-Info
GetAuthor	returns the developer of the DLL
GetDescription	returns what your DLL does
SetEasybotInfo	EasyBot calls this method of your class. As parameter you get an object of the class `Easybot_Interfaces`. It can be used to access all controls currently in use by Easybot.
SetRayInfo	EasyBot calls this method of your class. As parameter you get an array of `easybot_ray_info`-structures. They contain the sensor-information.
SetObjectInfo	EasyBot calls this method of your class. As parameter you get an element of the type `easybot_object_info`, that contains information about your robot.
SetMessageFrom	This method is triggered, when another robot `pfrom_robot` sends you a message `pmessage`.
GetMessageFor	EasyBot calls this method of your class just the same as `Navigate()`. EasyBot wants to know the robot (`**ppto_robot`), that receives the message (`**ppmessage`). The return-value tells EasyBot, whether you have a message or not. This method is called again, if it does not return 0.
Navigate	This method contains the control of the robot. When EasyBot executes it, all the sensor- and object-data has been set into the structures. It has to calculate the next step, using this data. You may use the operations “move” and “rotate”.
GetMoveX, GetMoveY, GetMoveZ	EasyBot calls this methods of your class, to get the relative movements of the robot in world-coordinates.
GetRotationX, GetRotationY, GetRotationZ	EasyBot calls this methods of your class, to get the rotation of the robot around the X, Y and Z-axis (in degree) in the object-coordinate system.
OpenDialog	If your control has its own dialog to interact with the user, that is the place to implement it. It can be used to parametrize the control.
SetStartStop	You get the information, that the simulation has started or stopped. (1=start, 0=stop)
Destroy	You should destroy your reserved resources, e.g. your modeless dialog.

The structures:

```
struct easybot_ray_info
{
   float   ray_length;     // length of ray
   float   ray_max_length; // max possible length of ray
   float   ray_sx;         // start-point (X,Y,Z)
   float   ray_sy;
   float   ray_sz;
   float   ray_ex;         // end-point (X,Y,Z)
   float   ray_ey;
   float   ray_ez;
   char*   obj_name;       // name of object hit by ray
   char*   mat_name;       // name of material hit by ray
   float   mat_clr_r;      // red-value of material
                           // (diffus component)
   float   mat_clr_g;      // green-value (0.0 .. 1.0)
   float   mat_clr_b;      // blue-value (0.0 .. 1.0)
   float   lgh_bright;     // brightness in ray direction
                           // on start-point
   float   lgh_r;          // light-red-value on start-point
   float   lgh_g;          // light-green-value (0.0 .. 1.0)
   float   lgh_b;          // light-blue-value (0.0 .. 1.0)
};
```

The structure `easybot_ray_info` provides information about a single ray. The meaning of each element is described in the comments. The information for the lights (`lgh_*`) is not currently used transferred. The `ray_max_length` is neither set, nor used by EasyBot. This data is free to be used for adjusting the ray.

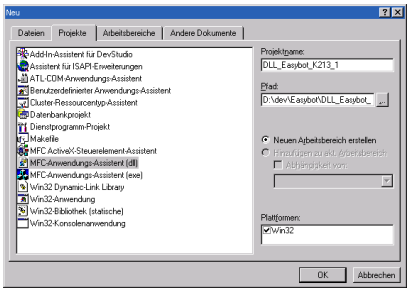
```
struct easybot_object_info
{
   float   obj_pos_x;  // position of the object in the
   float   obj_pos_y;  // world (X,Y,Z)
   float   obj_pos_z;
   float   obj_rot_x;  // rotation around the object-axis
   float   obj_rot_y;  // rotation-order: y,x,z
   float   obj_rot_z;  // y .. up, x .. right, z .. to you
   float   obj_go_x;   // direction the robot looks (goes)
   float   obj_go_y;
   float   obj_go_z;
   char*   obj_name;   // name of the robot
};
```

The structure `easybot_object_info` provides information about the robot. The meaning of each element is described in the comments. All elements are read-only!

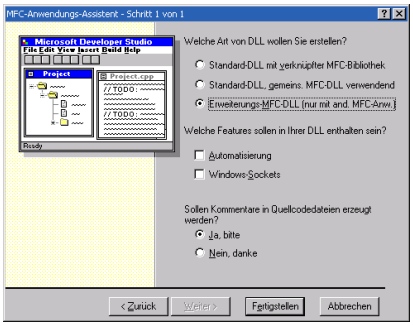
2.3.4 Lets build the DLL

Now we want to develop an own control-DLL.

1. Start the wizard, that constructs the framework for the DLL. Select “File/New”, choose the “MFC . . . (.dll)” frame-type, and give it the project-name “DLL_Khepera_K213_1”.



After “OK” we select the type of the DLL, as extension-DLL.



2. Now we define the external functions of our DLL. Therefore we open the `DLL_Khepera_K213_1.def` file, and write the name of the external function `GetEasybotInterface` into this file. `GetEasybotInterface` is the function, that calls EasyBot, to get an `DLL_Easybot_Interface`-object.

```
; DLL_Khepera_K213_1.def : declares module parametres for the DLL.

LIBRARY      "DLL_Khepera_K213_1"

DESCRIPTION  'DLL_Khepera_K213_1 Windows Dynamic Link Li-
brary'

EXPORTS
    ; explicite exports can be inserted here
        GetEasybotInterface
```

3. Define the external function `GetEasybotInterface` in the file `DLL_Khepera_K213_1.cpp` at the bottom of this file.

```
#include "Easybot_Khepera_K213_1.h"


DLL_Interface_Easybot* GetEasybotInterface()
{
   Easybot_Khepera_K213_1 *p;
   p=new Easybot_Khepera_K213_1;
   return p;
}
```

   This function returns a pointer to an object of our class.

4. Define the class `Easybot_Khepera_K213_1`, that inherits the `DLL_Easybot_Interface`. Save this file as `Easybot_Khepera_K213_1.h`, and add it to the project.

```
#ifndef EASYBOT_KHEPERA_K213_1
#define EASYBOT_KHEPERA_K213_1

#include "../include/easybot_interface.h"
#include "Dlg_K213.h"

class Easybot_Khepera_K213_1
                    :public DLL_Interface_Easybot
{
   float               m_fmovex,m_fmovey,m_fmovez,
                       m_frotx,m_froty,m_frotz;
   long                m_lNumberRays;
   int                 m_message,m_need_input,
                       m_last_speed,m_last_step,
                       m_last_rot;
   int                 m_rechtwinklig,m_dlg_open;
   CDlg_K213           *m_pdlg;
   easybot_ray_info    *pRays;
   easybot_object_info *pObject;
   Easybot_Interfaces  *pIntfs;
public:

   Easybot_Khepera_K213_1();
   ~Easybot_Khepera_K213_1();

   char* __cdecl GetName()
         {return "Easybot Khepera K213 1";}
```

```
   char* __cdecl GetVersion()
         {return "1.0";}
   char* __cdecl GetAuthor()
         {return "Oliver Michel";}
   char* __cdecl GetDescription()
     {return
     "navigation of a khepera with a K213 vision turret";}

   void  __cdecl SetEasybotInfo(Easybot_Interfaces *p)
         {pIntfs=p;}
   void  __cdecl SetRayInfo(easybot_ray_info *p,
                            long highest_index)
         {pRays=p;m_lNumberRays=highest_index+1;}
   void  __cdecl SetObjectInfo(easybot_object_info *p)
         {pObject=p;}

   void  __cdecl SetMessageFrom(char *pfrom_robot,
                                char *pmessage);
   int   __cdecl GetMessageFor(char **ppto_robot,
                               char **ppmessage);

   void  __cdecl Navigate();

   float __cdecl GetMoveX();
   float __cdecl GetMoveY();
   float __cdecl GetMoveZ();
   float __cdecl GetRotationX();
   float __cdecl GetRotationY();
   float __cdecl GetRotationZ();

   void  __cdecl OpenDialog();
   void  __cdecl SetStartStop(int onoff){}

   void  __cdecl Destroy(){delete this;}
};
#endif
```

5. Now we add the contents of the functions in `Easybot_Khepera_K213_1.cpp`.

```
#include "Easybot_Khepera_K213_1.h"

#define PIXEL_ROTATION(A,B) if \
   if ((pRays[A].mat_clr_r+pRays[A].mat_clr_g       \
                          +pRays[A].mat_clr_b)>2.6) \
   {y_intens+=B;see=1;}
```

```
Easybot_Khepera_K213_1::Easybot_Khepera_K213_1()
{
   m_fmovex=m_fmovey=m_fmovez=0;
   m_frotx=m_froty=m_frotz=0;
   m_message=1;
   m_need_input=0;
   m_rechtwinklig=0;
   m_pdlg=0;
   m_dlg_open=0;
}


Easybot_Khepera_K213_1::~Easybot_Khepera_K213_1()
{
   if (m_dlg_open)
      m_pdlg->DestroyWindow();
}


void    Easybot_Khepera_K213_1::SetMessageFrom
           (char *pfrom_robot, char *pmessage)
{
   // don't want to receive a messages
}


int     Easybot_Khepera_K213_1::GetMessageFor
           (char **ppto_robot, char **ppmessage)
{
   return 0;       // no messages
}


void    Easybot_Khepera_K213_1::Navigate()
{

   float   z_intens,y_intens,max_intens,old_y;
   int     see;

   max_intens=1;   //scope of the sensors
   z_intens=0.1f;  //move forward if no input
   y_intens=0;     //don't rotate if no input
```

```
// sensors of the base Khepera

if (pRays[0].ray_length<max_intens
 && pRays[0].ray_length>0)
{
   z_intens-=0.1f*((max_intens
                    -pRays[0].ray_length)/max_intens);
   y_intens-=2*((max_intens
                 -pRays[0].ray_length)/max_intens);
}

if (pRays[1].ray_length<max_intens
 && pRays[1].ray_length>0)
{
   z_intens-=0.1f*((max_intens
                    -pRays[1].ray_length)/max_intens);
   y_intens-=2*((max_intens
                 -pRays[1].ray_length)/max_intens);
}

if (pRays[2].ray_length<max_intens
 && pRays[2].ray_length>0)
{
   z_intens-=0.1f*((max_intens
                    -pRays[2].ray_length)/max_intens);
   y_intens-=2*((max_intens
                 -pRays[2].ray_length)/max_intens);
}

if (pRays[3].ray_length<max_intens
 && pRays[3].ray_length>0)
{
   z_intens-=0.1f*((max_intens
                    -pRays[3].ray_length)/max_intens);
   y_intens+=2*((max_intens
                 -pRays[3].ray_length)/max_intens);
}

if (pRays[4].ray_length<max_intens
 && pRays[4].ray_length>0)
{
   z_intens-=0.1f*((max_intens
                    -pRays[4].ray_length)/max_intens);
   y_intens+=2*((max_intens
                 -pRays[4].ray_length)/max_intens);
}
```

```
if (pRays[5].ray_length<max_intens
 && pRays[5].ray_length>0)
{
   z_intens-=0.1f*((max_intens
                    -pRays[5].ray_length)/max_intens);
   y_intens+=2*((max_intens
                 -pRays[5].ray_length)/max_intens);
}

if (pRays[6].ray_length<max_intens
 && pRays[6].ray_length>0)
{
   z_intens+=0.1f*((max_intens
                    -pRays[6].ray_length)/max_intens);
   y_intens-=0.5f*((max_intens
                    -pRays[6].ray_length)/max_intens);
}

if (pRays[7].ray_length<max_intens
 && pRays[7].ray_length>0)
{
   z_intens+=0.1f*((max_intens
                    -pRays[7].ray_length)/max_intens);
   y_intens+=0.5f*((max_intens
                    -pRays[7].ray_length)/max_intens);
}

// K213 vision turret

see=0;

PIXEL_ROTATION(8+30, 0.05);
PIXEL_ROTATION(8+32,-0.05);
PIXEL_ROTATION(8+23, 0.5);
PIXEL_ROTATION(8+39,-0.5);
if ((!see && m_rechtwinklig) || !m_rechtwinklig)
   // if option "rechtwinklige Ecken ausfahren"
   // active move forward along the line even
   // if there are other lines to be seen
{
   PIXEL_ROTATION(8+0 , 4);
   // ignore non-central input, as long as there
   // is a central input
   PIXEL_ROTATION(8+4 , 3);
   PIXEL_ROTATION(8+7 , 2);
```

```
      PIXEL_ROTATION(8+15, 1);
      PIXEL_ROTATION(8+47,-1);
      PIXEL_ROTATION(8+55,-2);
      PIXEL_ROTATION(8+59,-3);
      PIXEL_ROTATION(8+63,-4);
   }

   if (see) // if there is a line as non-central input,
            // rotate the robot to get it central
   {
      m_need_input=1;
      m_last_speed=z_intens;
      m_last_step=0.0;
      m_last_rot=0.0;
   }
   else
   {
      if (m_need_input)
      // robot left a line
      {
         // move further for the length of view radius
         if (m_last_step<14.0)
         {
            z_intens=0.1;
            m_last_step+=1.0;
         }
         else
            // we are at the end of the line, look for
            // another line by rotating +/- 90 degree
         {
            if (m_last_rot<90.0)
               // look (rotate) left
            {
               z_intens=0;
               y_intens=4;
               m_last_rot+=4.0;
            }
            else
            {
               if (m_last_rot<270.0)
                  // look (rotate) right
               {
                  z_intens=0;
                  y_intens=-4;
                  m_last_rot+=4.0;
               }
```

```
                else
                   // rotate back and go ahead
                {
                   if (m_last_rot<360.0)
                      // rotate back to
                      // original direction
                   {
                      z_intens=0;
                      y_intens=4;
                      m_last_rot+=4.0;
                   }
                   else
                      m_need_input=0;
                      // there was no line
                }
             }
          }
       }

       // Z-Richtung holen
       m_fmovex=pObject->obj_go_x;
       m_fmovey=pObject->obj_go_y;
       m_fmovez=pObject->obj_go_z;

       m_fmovex*=z_intens;
       m_fmovey*=z_intens;
       m_fmovez*=z_intens;

       m_froty=y_intens;

       if (m_dlg_open)
          m_pdlg->ShowK213();

       m_message=1;            // send another message
    }

    float   Easybot_Khepera_K213_1::GetMoveX()
    {
       return m_fmovex;
    }

    float   Easybot_Khepera_K213_1::GetMoveY()
    {
       return m_fmovey;
    }
```

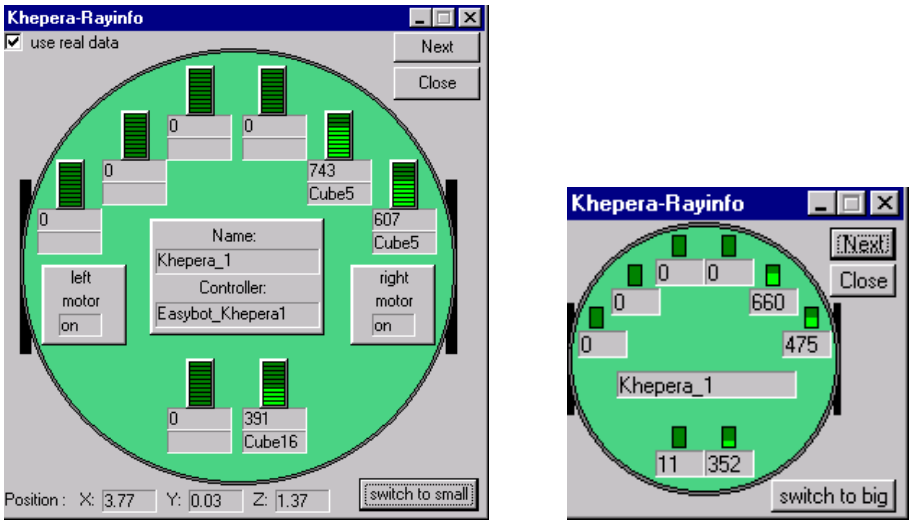
```
float   Easybot_Khepera_K213_1::GetMoveZ()
{
   return m_fmovez;
}

float   Easybot_Khepera_K213_1::GetRotationX()
{
   return m_frotx;
}

float   Easybot_Khepera_K213_1::GetRotationY()
{
   return m_froty;
}

float   Easybot_Khepera_K213_1::GetRotationZ()
{
   return m_frotz;
}

void    Easybot_Khepera_K213_1::OpenDialog()
{
   if (m_dlg_open==0)
   {
      if (!m_pdlg) m_pdlg=new CDlg_K213;
      m_pdlg->pRays=pRays;
      m_pdlg->m_popen=&m_dlg_open;
      m_pdlg->m_prechtwinklig=&m_rechtwinklig;
      m_pdlg->Create(IDD_DLG_K213);
      m_pdlg->ShowWindow(SW_SHOW);
      m_dlg_open=1;
   }
}
```

This is the general way to define a control-DLL. The special task is not explained here.

You can start testing the functionality using empty methods. Later you can add functionality step by step.

2.4 Manual for DLL_Khepera_Rayinfo1.dll

2.4.1 What's This ? – Description



(a) Big Dialog Panel (standard)

(b) Small Dialog Panel

Figure 2.2: Dialogs of the Easybot-Rayinfo

This is a control-DLL for EasyBot. It displays some information about Khepera-robots (most of them about distances and names of objects). The information is visualized by a dialog-panel. There is a circle with 8 groups of sensor-related textboxes and bitmaps to be seen. In the middle of the circle are the name- and controller-info of the Khepera.

The sensor-related group displays the data received by the corresponding sensor of the Khepera-robot in the simulation program. On top in this group is a bitmap which visualizes the proximity to an object in the simulator world. The more bars of the box are lit the closer the object is. Below these bitmaps are two textboxes. The upper one indicates either data of sensor-output like the real Khepera would receive (means numbers ranging from 0 up to 1024, 0 is far), or data from the simulator using the length of a vector. Actually the maximum length is 1.5 units[1]. If an object is further away it is no more recognized by the sensor. In case of simulated data 1.0e6 units indicate that there is no object in the range of the sensor. A value of 1.5 is far away and 0.0 means the object is hit. The lower textbox contains the name of the object "seen" by the sensor.

[1] a unit in LV3D is approx. 10 centimetres(cm)

There are two other textboxes in the centre of the circle. One labeled “Name:” contains the name of the Khepera-robot. The other one labeled “Controller” displays the name of the robot-navigation-module (DLL).

2.4.2 What can I do ? – Usage

To get to the panel:

1. start LV3D
2. load or create a scene with Khepera-robots
3. create an object (simplest is a triangle)
4. rename it to “Sensor”
   - object properties [click right mouse button on the object in the object browser]
   - “umbenennen...”
5. make it invisible
   - object properties
   - “Darstellung”
   - uncheck “aktiv”
6. start the robot-simulator
   - menu “Funktionen”
   - “Robotersimulation”
   - “EasyBot”
7. Load the proper navigation-modules for the Khepera-robots and probably other moving objects and load the “DLL_Khepera_Rayinfo1.dll” for the invisible object (supposedly “Sensor”).
8. “open dialog” for “Sensor” and you finally get the panel.

The upper section of the dialog-panel contains a checkbox and two buttons. The button named “Next” switches to the next robot that has a controll_DLL assigned to display its sensor-data. Reaching the bottom of the list it continues on top. This panel is especially designed for Khepera-robots with their typical 8-sensor-configuration. Sensor-data is only displayed for these robots. Other robots like doors, rotating doors, stand-alone sensors and so on will only have their names and controll_DLL displayed (The panel does not fit their sensor-data).

The button “Close” will close the Rayinfo-Dialog-panel.

The checkbox labeled “use real data” is set by default. That means the panel displays the sensor-data in the range from 0 up to 1024 like the real Khepera-sensors. An output of 1024 means the object is directly in front of the sensor.

If this checkbox is unchecked it displays simulated data. This is the distance between the sensor and an object (the length of the plane normal hitting an object) in LV3D-distance units. 0.0 units mean contact with the object, 1.5 units are the maximum, but the object is still noticed and 1.0e6 units mean there is nothing in sensor-range.

2.4.3 How does it work ? – Realization

This is a control-DLL for Easybot, that displays data about other connected robot-controllers. There are two dialog-panels implemented to achieve this.

To export the Rayinfo-Interface to EasyBot, it has to be declared in the file DLL_Khepera_Rayinfo1.def in section EXPORTS.

File DLL_Khepera_Rayinfo.def :

```
; DLL_Khepera_Rayinfo1.def : declares modul-parameters for the DLL.

LIBRARY      "DLL_Khepera_Rayinfo1"
DESCRIPTION  'DLL_Khepera_Rayinfo1 Windows Dynamic Link Li-
brary'

EXPORTS
    ; Explicit exports can go here
        GetEasybotInterface
```

This declaration is necessary to link the control-DLL to EasyBot. The function is defined as follows:

```
DLL_Interface_Easybot* GetEasybotInterface()
{
        Easybot_Rayinfo *p;
        p=new Easybot_Rayinfo;
        return p;
}
```

This function creates a new Easybot_Rayinfo-object. The class is defined in the files Easybot_Rayinfo.h and Easybot_Rayinfo.cpp:

```
// Easybot_Rayinfo.h
// Headerfile for Rayinfo-Easybot
// Marko Hassler - htw3566
```

```
#ifndef EASYBOT_RAYINFO
#define EASYBOT_RAYINFO

#include "../include/easybot_interface.h"
#include "Dlg_Rayinfo.h"
#include "Dlg_Small_Rayinfo.h"


class Easybot_Rayinfo:public DLL_Interface_Easybot
{
   int                    m_dlg_state;
                          // state-variable - this is a global
                          // variable to sync to dialogs
   int                    n, count; // simple counter
   long                   m_lNumberRays; // for SetRayInfo()
   long                   curr_num_rays; // holds the number
                          //  of rays of current easybot
   CDlg_Rayinfo           *m_pdlg; // large Dialog
   CDlg_Small_Rayinfo     *m_pdlgsm; // small Dialog
   easybot_ray_info       *pRays; // rayinfo ray(s) := none
   easybot_object_info    *pObject; // rayinfo object
   Easybot_Interfaces     *pIntfs; // rayinfo interface

   easybot_ray_info       *pCurrRays; // current rays
   easybot_object_info    *pCurrObject; // current object
   DLL_Interface_Easybot  *pCurrDLL; // current DLL_Interface
   DLL_Interface_Easybot  *pTry; // test DLL_Interface

   CString   sensor_test; // test-string for khepera-names

   // the arrays allow indexed processing
   CStatic   *m_bild[8]; // Array of pointer to CStatic-controls
   CString   *m_hit[8]; // Array of pointer to CString-objects
   CString   *m_sensor[8]; // Array of pointer to Cstring-
objects

   CStatic   *m_bildsm[8]; // Array of pointer to CStatic-
controls
   CString   *m_sensorsm[8]; // Array of pointer to Cstring-
objects

public:

   Easybot_Rayinfo();
   ~Easybot_Rayinfo();
```

```
char*   __cdecl GetName(){return "Easybot Rayinfo";}
char*   __cdecl GetVersion(){return "1.0";}
char*   __cdecl GetAuthor(){return "Marko Hassler";}
char*   __cdecl GetDescription()
     {return "display of information from sensors";}

void    __cdecl SetInterfaceInfo(Easybot_Interfaces *p)
     {pIntfs=p;}
void    __cdecl SetRayInfo(easybot_ray_info *p,
                           long highest_index)
     {pRays=p;m_lNumberRays=highest_index+1;}
void    __cdecl SetObjectInfo(easybot_object_info *p)
     {pObject=p;}

void    __cdecl SetMessageFrom(char *pfrom_robot,
                               char *pmessage){}
     // get messages from other robots - no message
int     __cdecl GetMessageFor(char **ppto_robot,
                              char **ppmessage)
     {return 0;}
     // send messages to other robots
     // (return 1: valid message, 0: no more messages)

void    __cdecl Navigate();
     // next navigation step - most important

double  __cdecl GetMoveX(){return 0.0;}
     // relative move in X (world-coords) -none-
double  __cdecl GetMoveY(){return 0.0;}
     // relative move in Y (world-coords) -none-
double  __cdecl GetMoveZ(){return 0.0;}
     // relative move in Z (world-coords) -none-
double  __cdecl GetRotationX(){return 0.0;}
     // rotation around X-axis in degree -none-
double  __cdecl GetRotationY(){return 0.0;}
     // rotation around Y-axis in degree -none-
double  __cdecl GetRotationZ(){return 0.0;}
     // rotation around Z-axis in degree -none-

void    __cdecl OpenDialog(); //  dito

void    __cdecl SetStartStop(int onoff){}
     //  not used

void    __cdecl Destroy(){delete this;}
     //  destroy Easybot-object
```

```
};

#endif

// Easybot_Rayinfo.cpp
// Sourcefile for Rayinfo-Easybot
// Marko Hassler - htw3566

#include "StdAfx.h"
#include "Easybot_Rayinfo.h"
#include <math.h>


Easybot_Rayinfo::Easybot_Rayinfo()
{
   m_pdlg=0; // no large dialog, yet
   m_pdlgsm=0; // no small dialog too
   m_dlg_state=0; // state variable
   count=1; // process first robot in the beginning
}


Easybot_Rayinfo::~Easybot_Rayinfo()
{
   // cleanup the dialogs
   if (m_pdlg)
      m_pdlg->DestroyWindow();
   if (m_pdlgsm)
      m_pdlgsm->DestroyWindow();
}


void    Easybot_Rayinfo::Navigate()
{
   if (m_dlg_state==0)
      return; // no dialog open, so do nothing

   if (m_dlg_state==2) // "switch to small" was pressed ?
   {
      m_pdlg->ShowWindow(SW_HIDE); // hide large dialog
      m_pdlgsm->ShowWindow(SW_SHOW); // show small dialog
   }

   if (m_dlg_state==3) // "switch to big" was pressed ?
   {
      m_pdlgsm->ShowWindow(SW_HIDE); // hide small dialog
```

```
   m_pdlg->ShowWindow(SW_SHOW); // show large dialog
   m_dlg_state=1;
       // reset state variable to "normal" condition
}

// get proper robot data:
if (m_pdlg->m_next || m_pdlgsm->m_next) // next button
   // pressed ? - then take next Khepera-robot in list
{
   count=0;  // deactivate the routine for first robot

   /* Pointer to Interface is local for this Easybot.
      So it has to be reset each time navigate is
      called. Luckily, we know which robot is current
      because of data saved in pCurrDLL.
   */
   pTry=pIntfs->GetFirstInterface();
   while(pTry)                 // go to current robot
   {
      if (pTry==pCurrDLL)
         break;
      pTry=pIntfs->GetNextInterface();
   }

   if ( (pCurrDLL = pIntfs->GetNextInterface()) == 0 )
                     // no more robots?
      count=1;       // take first robot
   else              // take next robot in the list
   {
                     // assign corresponding infos
      pCurrObject = pIntfs->GetObjectInfo();
      pCurrRays = pIntfs->GetRayInfo();
      curr_num_rays = pIntfs->GetNumberOfRays();
      sensor_test=pCurrObject->obj_name;

      while ( (sensor_test.Find("Khepera") == -1)
           && (sensor_test.Find("khepera") == -1)
           && (sensor_test.Find("K213") == -1)
           && (curr_num_rays != 8)
            ) // while no Khepera
      {
         if ((pCurrDLL=pIntfs->GetNextInterface()) == 0)
                           // no more robots?
         {
            count=1;  // take the first robot
            break;    // leave while loop
```

```
            }
                        // take this robot,
                        // it is the next available Khepera
            pCurrObject = pIntfs->GetObjectInfo();
            pCurrRays = pIntfs->GetRayInfo();
            curr_num_rays = pIntfs->GetNumberOfRays();
            sensor_test=pCurrObject->obj_name;
         } // endwhile
      } // endelse next robot

      m_pdlg->m_next=0; // reset next buttons
      m_pdlgsm->m_next=0;
   }

   if (count) //  first robot in the list
   {
      // first interface and corresponding infos
      pCurrDLL = pIntfs->GetFirstInterface();
      pCurrObject = pIntfs->GetObjectInfo();
      pCurrRays = pIntfs->GetRayInfo();
      curr_num_rays = pIntfs->GetNumberOfRays();
      sensor_test=pCurrObject->obj_name;

      while ( (sensor_test.Find("Khepera") == -1)
           && (sensor_test.Find("khepera") == -1)
           && (sensor_test.Find("K213") == -1)
           && (curr_num_rays != 8)
            ) // while this is no Khepera
      {
         if ((pCurrDLL=pIntfs->GetNextInterface()) == 0)
                           // no other robot ?
         {
                           // fill in dummy data
            m_pdlg->m_name.Format("no robot");
            m_pdlgsm->m_smname.Format("no robot");
            m_pdlg->m_steuerung.Format("assign controller DLL");
            m_pdlg->m_leftmotor.Format("off");
            m_pdlg->m_rightmotor.Format("off");
            for (n=0;n<8;n++)
            {
                // Bitmaps
               m_bild[n]->SetBitmap(m_pdlg->bm[0]);
               m_bildsm[n]->SetBitmap(m_pdlgsm->bm_sm[0]);
                // object info textboxes
               m_sensor[n]->Format("inactive");
               m_sensorsm[n]->Format("off");
```

```
                // sensor info testboxes
               m_hit[n]->Format("inactive");
            }
            m_pdlg->UpdateData(false);
            m_pdlgsm->UpdateData(false);
            return;      // failure, nothing to display,
                    // request user for DLLs
         }

                       // take this robot,
                       // it is the first Khepera in the list
         pCurrObject = pIntfs->GetObjectInfo();
         pCurrRays = pIntfs->GetRayInfo();
         curr_num_rays = pIntfs->GetNumberOfRays();
         sensor_test=pCurrObject->obj_name;
      } // endwhile
   } // endif first robot


   //now fill the Dialog:
   //  set appropriate steuerungs-name
   m_pdlg->m_steuerung.Format("%s", pCurrDLL->GetName() );

   //      object-related information ( Name, Pos)
   m_pdlg->m_name.Format("%s", pCurrObject->obj_name);
   m_pdlgsm->m_smname.Format("%s", pCurrObject->obj_name);
   m_pdlg->m_x.Format("%.3g", pCurrObject->obj_pos_x);
   m_pdlg->m_y.Format("%.3g", pCurrObject->obj_pos_y);
   m_pdlg->m_z.Format("%.3g", pCurrObject->obj_pos_z);

   // dummy, data still needed
   m_pdlg->m_leftmotor.Format("on");
   // m_pdlg->m_leftmotor.Format("%i",
   //                      pCurrObject->left_motor_speed);
   // or calculated data for the speed of left motor

   // dummy, see directly below
   m_pdlg->m_rightmotor.Format("on");
   // m_pdlg->m_rightmotor.Format("%i",
   //                     pCurrObject->right_motor_speed);
   // or calculated data for the speed of right motor


   // display ray-related information (length of ray,
   // object hit) of sensor groups
```

```
   /*
   short notice :
   all calculations for the sensors were done, assuming
   that the largest range of the sensors is 1.5 LV3D-units
   (by now one unit := 10 centimetres)
   */

   for (n=0;n<8;n++) // do this for all 8 sensor groups
   {
      // Bitmaps for large and small dialog
      m_bild[n]->SetBitmap(
         m_pdlg->bm[pCurrRays[n].ray_length<=1.5?
      10-(int)floor(pCurrRays[n].ray_length/1.5*10+.5):0]);

      m_bildsm[n]->SetBitmap(
         m_pdlgsm->bm_sm[pCurrRays[n].ray_length<=1.5?
      10-(int)floor(pCurrRays[n].ray_length/1.5*10+.5):0]);

      // sensor info textboxes
      if (m_pdlg->m_modeldata)  //  fill in Khepera data
         m_sensor[n]->Format("%i",
            pCurrRays[n].ray_length<=1.5?
         1024-(int)floor(pCurrRays[n].ray_length/1.5*1024+.5):0);

     else                      //  fill in simulator data
         m_sensor[n]->Format("%.3g",
                             pCurrRays[n].ray_length<=1.5?
                             pCurrRays[n].ray_length:1e6);

         m_sensorsm[n]->Format("%i",
                               pCurrRays[n].ray_length<=1.5?
         1024-(int)floor(pCurrRays[n].ray_length/1.5*1024+.5):0);

      // object-hit-textboxes, this is only for large dialog
      m_hit[n]->Format("%s",
                       pCurrRays[n].ray_length<=1.5?
                       pCurrRays[n].obj_name:"");
   } // endfor sensor groups

   m_pdlg->UpdateData(false); // Force Update of dialog
   m_pdlgsm->UpdateData(false); // dito for small dialog
}


void    Easybot_Rayinfo::OpenDialog()
        // if "Open Dialog"-button is pressed
```

```
{
   if (m_dlg_state==0) // only act, when dialogs are off
   {
      if (m_pdlg) // is there already an large-dialog-object ?
         m_pdlg->DestroyWindow(); // trash it

      m_pdlg=new CDlg_Rayinfo; // allocate large dialog

        // bind pointer to state variable
      m_pdlg->m_popen=&m_dlg_state;
        // create modeless dialog
      m_pdlg->Create(IDD_RAYINFO_DLG);
        // show large dialog window
      m_pdlg->ShowWindow(SW_SHOW);

        // fill array of pointers to CStatics
        // for easier handling with for(;;)-loops
      m_bild[0]=&m_pdlg->m_bildS0;
      m_bild[1]=&m_pdlg->m_bildS1;
      m_bild[2]=&m_pdlg->m_bildS2;
      m_bild[3]=&m_pdlg->m_bildS3;
      m_bild[4]=&m_pdlg->m_bildS4;
      m_bild[5]=&m_pdlg->m_bildS5;
      m_bild[6]=&m_pdlg->m_bildS6;
      m_bild[7]=&m_pdlg->m_bildS7;

        // do the same with array of pointers to CStrings
      m_sensor[0]=&m_pdlg->m_sensor0;
      m_sensor[1]=&m_pdlg->m_sensor1;
      m_sensor[2]=&m_pdlg->m_sensor2;
      m_sensor[3]=&m_pdlg->m_sensor3;
      m_sensor[4]=&m_pdlg->m_sensor4;
      m_sensor[5]=&m_pdlg->m_sensor5;
      m_sensor[6]=&m_pdlg->m_sensor6;
      m_sensor[7]=&m_pdlg->m_sensor7;
      m_hit[0]=&m_pdlg->m_hitobject0;
      m_hit[1]=&m_pdlg->m_hitobject1;
      m_hit[2]=&m_pdlg->m_hitobject2;
      m_hit[3]=&m_pdlg->m_hitobject3;
      m_hit[4]=&m_pdlg->m_hitobject4;
      m_hit[5]=&m_pdlg->m_hitobject5;
      m_hit[6]=&m_pdlg->m_hitobject6;
      m_hit[7]=&m_pdlg->m_hitobject7;

      if (m_pdlgsm)
         // is there already an small-dialog-object ?
```

```
         m_pdlgsm->DestroyWindow(); // trash it

         // allocate small dialog
      m_pdlgsm=new CDlg_Small_Rayinfo;

         // bind this pointer to state variable too
      m_pdlgsm->m_popen=&m_dlg_state;

         // create modeless dialog
      m_pdlgsm->Create(IDD_SMALLRAY_DLG);
         // hide this small dialog window at first
      m_pdlgsm->ShowWindow(SW_HIDE);

         // fill arrays for handling with for(;;)-loops
      m_bildsm[0]=&m_pdlgsm->m_smbildS0;
      m_bildsm[1]=&m_pdlgsm->m_smbildS1;
      m_bildsm[2]=&m_pdlgsm->m_smbildS2;
      m_bildsm[3]=&m_pdlgsm->m_smbildS3;
      m_bildsm[4]=&m_pdlgsm->m_smbildS4;
      m_bildsm[5]=&m_pdlgsm->m_smbildS5;
      m_bildsm[6]=&m_pdlgsm->m_smbildS6;
      m_bildsm[7]=&m_pdlgsm->m_smbildS7;
      m_sensorsm[0]=&m_pdlgsm->m_smsensor0;
      m_sensorsm[1]=&m_pdlgsm->m_smsensor1;
      m_sensorsm[2]=&m_pdlgsm->m_smsensor2;
      m_sensorsm[3]=&m_pdlgsm->m_smsensor3;
      m_sensorsm[4]=&m_pdlgsm->m_smsensor4;
      m_sensorsm[5]=&m_pdlgsm->m_smsensor5;
      m_sensorsm[6]=&m_pdlgsm->m_smsensor6;
      m_sensorsm[7]=&m_pdlgsm->m_smsensor7;

         // state variable is set to: dialogs open,
         // large is shown, small is hidden
      m_dlg_state=1;
   }
}
```

The control-DLL described above, gets data about other controlled objects. This data is shown on a dialog panel. The source code for the large panel:

```
#ifndef AFX_DLG_RAYINFO_H__INCLUDED_
#define AFX_DLG_RAYINFO_H__INCLUDED_

// Dlg_Rayinfo.h :
// Header-File for Rayinfo-Dialog
// Marko Hassler - htw3566
```

```
#include "resource.h"
#include "StdAfx.h"

/////////////////////////////////////////////////////

class CDlg_Rayinfo : public CDialog
{
// Konstruktion
public:
   CDlg_Rayinfo(CWnd* pParent = NULL);

   int      *m_popen; // (large) dialog open ?
   int       m_next; // next-button pressed ?

   HBITMAP   bm[11]; // Bitmaps for values from 0..10


   //{{AFX_DATA(CDlg_Rayinfo)
   enum { IDD = IDD_RAYINFO_DLG };
   CStatic m_bildS7;
   CStatic m_bildS6;
   CStatic m_bildS5;
   CStatic m_bildS4;
   CStatic m_bildS3;
   CStatic m_bildS2;
   CStatic m_bildS1;
   CStatic m_bildS0;
   CString m_x;
   CString m_y;
   CString m_z;
   CString m_hitobject0;
   CString m_hitobject1;
   CString m_hitobject2;
   CString m_hitobject3;
   CString m_hitobject4;
   CString m_hitobject5;
   CString m_hitobject6;
   CString m_hitobject7;
   CString m_name;
   CString m_steuerung;
   CString m_sensor0;
   CString m_sensor1;
   CString m_sensor2;
   CString m_sensor3;
   CString m_sensor4;
```

```
   CString m_sensor5;
   CString m_sensor6;
   CString m_sensor7;
   BOOL    m_modeldata;
   CString m_leftmotor;
   CString m_rightmotor;
   //}}AFX_DATA


   //{{AFX_VIRTUAL(CDlg_Rayinfo)
   protected:
   virtual void DoDataExchange(CDataExchange* pDX);
   //}}AFX_VIRTUAL

// Implementation
protected:

   //{{AFX_MSG(CDlg_Rayinfo)
   virtual void OnCancel();
   afx_msg void OnCheckModeldata();
   afx_msg void OnNaechster();
   virtual BOOL OnInitDialog();
   afx_msg void OnSwitchSmall();
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}


#endif // AFX_DLG_RAYINFO_H__INCLUDED_

// Dlg_Rayinfo.cpp
// Implementation file for Rayinfo-Dialog
// Marko Hassler - htw3566

#include "stdafx.h"
#include "Dlg_Rayinfo.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////
```

```
CDlg_Rayinfo::CDlg_Rayinfo(CWnd* pParent /*=NULL*/)
   : CDialog(CDlg_Rayinfo::IDD, pParent)
{
   //{{AFX_DATA_INIT(CDlg_Rayinfo)
   m_x = _T("");
   m_y = _T("");
   m_z = _T("");
   m_hitobject0 = _T("");
   m_hitobject1 = _T("");
   m_hitobject2 = _T("");
   m_hitobject3 = _T("");
   m_hitobject4 = _T("");
   m_hitobject5 = _T("");
   m_hitobject6 = _T("");
   m_hitobject7 = _T("");
   m_name = _T("");
   m_steuerung = _T("");
   m_sensor0 = _T("");
   m_sensor1 = _T("");
   m_sensor2 = _T("");
   m_sensor3 = _T("");
   m_sensor4 = _T("");
   m_sensor5 = _T("");
   m_sensor6 = _T("");
   m_sensor7 = _T("");
   m_modeldata = FALSE;
   m_leftmotor = _T("");
   m_rightmotor = _T("");
   //}}AFX_DATA_INIT
   m_popen=0; // set state-variable to 0
   m_next=0; // initialize next-button to not-pressed
   // "use real data" is checked, values from 0..1024
   m_modeldata=TRUE;
}


void CDlg_Rayinfo::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CDlg_Rayinfo)
   DDX_Control(pDX, IDC_GRA_S7, m_bildS7);
   DDX_Control(pDX, IDC_GRA_S6, m_bildS6);
   DDX_Control(pDX, IDC_GRA_S5, m_bildS5);
   DDX_Control(pDX, IDC_GRA_S4, m_bildS4);
```

```
   DDX_Control(pDX, IDC_GRA_S3, m_bildS3);
   DDX_Control(pDX, IDC_GRA_S2, m_bildS2);
   DDX_Control(pDX, IDC_GRA_S1, m_bildS1);
   DDX_Control(pDX, IDC_GRA_S0, m_bildS0);
   DDX_Text(pDX, IDC_X, m_x);
   DDX_Text(pDX, IDC_Y, m_y);
   DDX_Text(pDX, IDC_Z, m_z);
   DDX_Text(pDX, IDC_HIT0, m_hitobject0);
   DDX_Text(pDX, IDC_HIT1, m_hitobject1);
   DDX_Text(pDX, IDC_HIT2, m_hitobject2);
   DDX_Text(pDX, IDC_HIT3, m_hitobject3);
   DDX_Text(pDX, IDC_HIT4, m_hitobject4);
   DDX_Text(pDX, IDC_HIT5, m_hitobject5);
   DDX_Text(pDX, IDC_HIT6, m_hitobject6);
   DDX_Text(pDX, IDC_HIT7, m_hitobject7);
   DDX_Text(pDX, IDC_NAME, m_name);
   DDX_Text(pDX, IDC_STEUERUNG, m_steuerung);
   DDX_Text(pDX, IDC_SENSOR0, m_sensor0);
   DDX_Text(pDX, IDC_SENSOR1, m_sensor1);
   DDX_Text(pDX, IDC_SENSOR2, m_sensor2);
   DDX_Text(pDX, IDC_SENSOR3, m_sensor3);
   DDX_Text(pDX, IDC_SENSOR4, m_sensor4);
   DDX_Text(pDX, IDC_SENSOR5, m_sensor5);
   DDX_Text(pDX, IDC_SENSOR6, m_sensor6);
   DDX_Text(pDX, IDC_SENSOR7, m_sensor7);
   DDX_Check(pDX, IDC_CHECK_MODELDATA, m_modeldata);
   DDX_Text(pDX, IDC_LEFTMOTOR, m_leftmotor);
   DDX_Text(pDX, IDC_RIGHTMOTOR, m_rightmotor);
   //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDlg_Rayinfo, CDialog)
   //{{AFX_MSG_MAP(CDlg_Rayinfo)
   ON_BN_CLICKED(IDC_CHECK_MODELDATA, OnCheckModeldata)
   ON_BN_CLICKED(IDC_NAECHSTER, OnNaechster)
   ON_BN_CLICKED(IDC_SWITCHSMALL, OnSwitchSmall)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////

void CDlg_Rayinfo::OnCancel()
{
   *m_popen=0; // set state variable to 0 -> dialogs are closed
   DestroyWindow(); // dito -> cleanup for modeless dialogs
```

```
}

BOOL CDlg_Rayinfo::OnInitDialog()
{
   CDialog::OnInitDialog();

   // create and fill an array of bitmaps
   // for easier processing (indexing)
   int bmp[11] = { IDB_LARGE0, IDB_LARGE1, IDB_LARGE2,
                   IDB_LARGE3, IDB_LARGE4, IDB_LARGE5,
                   IDB_LARGE6, IDB_LARGE7, IDB_LARGE8,
                   IDB_LARGE9, IDB_LARGE10 };

   for (int n=0;n<11;n++)
   {
      bm[n]=::LoadBitmap(AfxGetResourceHandle(),
               MAKEINTRESOURCE(bmp[n]));
   }

   return TRUE;
}

void CDlg_Rayinfo::OnCheckModeldata()
{
   UpdateData(true);
   // check/uncheck the box, switch variable m_modeldata
}

void CDlg_Rayinfo::OnNaechster()
{
   m_next=1;
   // next button was pressed,
   // processed in Navigate() of Easybot
}

void CDlg_Rayinfo::OnSwitchSmall()
{
   *m_popen=2;
   // set state variable to 2 -> close large dialog
   // and open small dialog in Navigate() of Easybot
}
```

And here for the small panel:

```
#ifndef AFX_DLG_SMALL_RAYINFO_H__INCLUDED_
#define AFX_DLG_SMALL_RAYINFO_H__INCLUDED_
```

```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

// Dlg_Small_Rayinfo.h :
// Header-Datei for Small Rayinfo Dialog
// Marko Hassler - htw3566

#include "resource.h"
#include "StdAfx.h"

/////////////////////////////////////////////////////////
// Dialogfeld CDlg_Small_Rayinfo

class CDlg_Small_Rayinfo : public CDialog
{
// Konstruktion
public:
   CDlg_Small_Rayinfo(CWnd* pParent = NULL);

   int       *m_popen; // (small) dialog open ?
   int        m_next;  // next-button pressed ?

   HBITMAP    bm_sm[11]; // Bitmaps for values from 0..10


   //{{AFX_DATA(CDlg_Small_Rayinfo)
   enum { IDD = IDD_SMALLRAY_DLG };
   CStatic m_smbildS7;
   CStatic m_smbildS6;
   CStatic m_smbildS5;
   CStatic m_smbildS4;
   CStatic m_smbildS3;
   CStatic m_smbildS2;
   CStatic m_smbildS0;
   CStatic m_smbildS1;
   CString m_smname;
   CString m_smsensor0;
   CString m_smsensor1;
   CString m_smsensor2;
   CString m_smsensor3;
   CString m_smsensor4;
   CString m_smsensor5;
   CString m_smsensor6;
   CString m_smsensor7;
```

```
   //}}AFX_DATA


   //{{AFX_VIRTUAL(CDlg_Small_Rayinfo)
   protected:
   virtual void DoDataExchange(CDataExchange* pDX);
   //}}AFX_VIRTUAL

// Implementation
protected:

   //{{AFX_MSG(CDlg_Small_Rayinfo)
   afx_msg void OnSwitchbig();
   virtual BOOL OnInitDialog();
   afx_msg void OnNaechster1();
   afx_msg void OnCancel1();
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}


#endif // AFX_DLG_SMALL_RAYINFO_H__INCLUDED_

// Dlg_Small_Rayinfo.cpp:
// Implementation file for Small-Rayinfo-Dialog
// Marko Hassler - htw3566

#include "stdafx.h"
#include "Dlg_Small_Rayinfo.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////


CDlg_Small_Rayinfo::CDlg_Small_Rayinfo(CWnd* pParent)
   : CDialog(CDlg_Small_Rayinfo::IDD, pParent)
{
   //{{AFX_DATA_INIT(CDlg_Small_Rayinfo)
   m_smname = _T("");
```

```
   m_smsensor0 = _T("");
   m_smsensor1 = _T("");
   m_smsensor2 = _T("");
   m_smsensor3 = _T("");
   m_smsensor4 = _T("");
   m_smsensor5 = _T("");
   m_smsensor6 = _T("");
   m_smsensor7 = _T("");
   //}}AFX_DATA_INIT
   m_popen=0; // set state-variable to 0
   m_next=0; // initialize next-button to not-pressed
}


void CDlg_Small_Rayinfo::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CDlg_Small_Rayinfo)
   DDX_Control(pDX, IDC_SMA_S7, m_smbildS7);
   DDX_Control(pDX, IDC_SMA_S6, m_smbildS6);
   DDX_Control(pDX, IDC_SMA_S5, m_smbildS5);
   DDX_Control(pDX, IDC_SMA_S4, m_smbildS4);
   DDX_Control(pDX, IDC_SMA_S3, m_smbildS3);
   DDX_Control(pDX, IDC_SMA_S2, m_smbildS2);
   DDX_Control(pDX, IDC_SMA_S0, m_smbildS0);
   DDX_Control(pDX, IDC_SMA_S1, m_smbildS1);
   DDX_Text(pDX, IDC_NAMESM, m_smname);
   DDX_Text(pDX, IDC_SENSORSM0, m_smsensor0);
   DDX_Text(pDX, IDC_SENSORSM1, m_smsensor1);
   DDX_Text(pDX, IDC_SENSORSM2, m_smsensor2);
   DDX_Text(pDX, IDC_SENSORSM3, m_smsensor3);
   DDX_Text(pDX, IDC_SENSORSM4, m_smsensor4);
   DDX_Text(pDX, IDC_SENSORSM5, m_smsensor5);
   DDX_Text(pDX, IDC_SENSORSM6, m_smsensor6);
   DDX_Text(pDX, IDC_SENSORSM7, m_smsensor7);
   //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDlg_Small_Rayinfo, CDialog)
   //{{AFX_MSG_MAP(CDlg_Small_Rayinfo)
   ON_BN_CLICKED(IDC_SWITCHBIG, OnSwitchbig)
   ON_BN_CLICKED(IDC_NAECHSTER1, OnNaechster1)
   ON_BN_CLICKED(IDCANCEL1, OnCancel1)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////////////////////

void CDlg_Small_Rayinfo::OnCancel1()
{
   // set state variable to 0 -> dialogs are closed
   *m_popen=0;
   // dito -> cleanup for modeless dialogs
   DestroyWindow();
}

BOOL CDlg_Small_Rayinfo::OnInitDialog()
{
   CDialog::OnInitDialog();

   // create and fill an array of bitmaps
   // for easier processing (indexing)
   int bmp_sm[11] = { IDB_SMALL0, IDB_SMALL1, IDB_SMALL2,
                      IDB_SMALL3, IDB_SMALL4, IDB_SMALL5,
                      IDB_SMALL6, IDB_SMALL7, IDB_SMALL8,
                      IDB_SMALL9, IDB_SMALL10 };

   for (int n=0;n<11;n++)
   {
      bm_sm[n]=::LoadBitmap(AfxGetResourceHandle(),
                  MAKEINTRESOURCE(bmp_sm[n]));
   }

   return TRUE;
}

void CDlg_Small_Rayinfo::OnNaechster1()
{
   m_next=1;
   // next button was pressed, processed
   // in Navigate() of Easybot
}

void CDlg_Small_Rayinfo::OnSwitchbig()
{
   *m_popen=3;
   // set state variable to 3 -> close small dialog
   // and open large dialog in Navigate() of Easybot
}
```

The data is acquired in the `Navigate()`-Routine of the Rayinfo-object. The visualization mode of the Rayinfo-object is managed by a variable (`m_dialog_state`), that can have these four states:

- State 0 - Both dialogs are closed.
- State 1 - Both dialogs are created. The large one is shown, the small is hidden.
- State 2 - "switch to small"-button has been pressed. When `Navigate()` is executed the next time, the large dialog will be hidden and the small will be shown.
- State 3 - "switch to big"-button has been pressed. At the next execution of the `Navigate()` routine of Easybot_Rayinfo the small dialog will be hidden and the large will be shown again. Then the `m_dialog_state` variable is reset to state 1 to avoid unnecessary calls of `ShowWindow()`.

The reason for using arrays in the program was to structure the source code with the help of `for(...)` loops and indices. As there are 8 similar sensor groups with 3 elements each (for the large dialog) it would be harder to read such code if it were written straight forward. The counting of the sensors begins on the left with number 0, continues clockwise and ends with 7 at the left back-sensor. This numbering was done equally as the real Khepera-robot-sensors are numbered. The data is updated each time `Navigate()` is called by EasyBot. Then the bitmaps and the strings of the textboxes are set according to the current values of the monitored robot.

2.4.4 What's left to do ? – Future Plans

The Rayinfo-panel should be integrated in the EasyBot-dialog to avoid the complicated connection process of the Rayinfo-module with virtual dummy sensors. Then there will be just a checkbox to activate this panel.

The motor speeds could be shown in the two textboxes which are already integrated in the panel. The textboxes are connected with the Rayinfo_Easybot. What is left to do, is to provide this information and link it.

2.5 Programming of Extension Modules with Delphi

The Easybot interface was designed to be able to write extension modules in any language, which support C-style function calls. So you can also write them in Delphi.

Why should one want to use Delphi? There are several reasons. One is almost everyone who starts to learn a programming language starts with Pascal, except maybe computer science students which start with C. Another important aspect is, how easy you can build a graphical interface for your program, and here is Delphi much easier than using the MFC.

So Delphi is ideal for writing modules by beginners or if you want to test a new control algorithm, which needs a graphical interface.

2.5.1 The EasyBot Interface in Delphi

Now we have to translate all data types to Pascal, which is relatively easy because for every C++ data type there is a corresponding one in Delphi. For char-pointer we have in Delphi the special type PChar. To make it easier we also define data types for pointers to some important types like easybot_object_info and so on. In addition we need for easybot_ray_info an array type, because the parameter of the function SetRayInfo() was specially designed to be compatible with Delphi (which internally uses a pointer to the array and the highest index).

The other aspect is that all functions have to be *virtual* and use the calling convention *cdecl*. This guarantees that there is a compatible *vtable* in the object.

An other point which is to consider is the function *Destroy()*, which should be renamed because Delphi uses this name for destructors. So we change this to *DestroyInterface()*.

And now let us have a look at the code (file: UDLL_Interface_Easybot.pas).

```
unit UDLL_Interface_Easybot;

interface

type
  easybot_ray_info=RECORD
    ray_length:double;     // ray length between start and end
    ray_max_length:double; // max possible length of the ray
    ray_sx:double;         // start-point (X,Y,Z)
    ray_sy:double;
    ray_sz:double;
    ray_ex:double;         // end-point (X,Y,Z)
    ray_ey:double;
    ray_ez:double;
    obj_name:PChar;        // name of object hit by ray
    mat_name:PChar;        // name of material hit by ray
```

```
    mat_clr_r:double;      // diffuse red-value of material
    mat_clr_g:double;      // green-value (0.0 .. 1.0)
    mat_clr_b:double;      // blue-value (0.0 .. 1.0)
    lgh_bright:double;     // brightness at start-point
    lgh_r:double;          // light-red-value at start-point
    lgh_g:double;          // light-green-value (0.0 .. 1.0)
    lgh_b:double;          // light-blue-value (0.0 .. 1.0)
  end;

  // a pointer to a record
  Peasybot_ray_info=^easybot_ray_info;

  // an array of records
  Aeasybot_ray_info= array of easybot_ray_info;

  easybot_object_info=RECORD
    obj_pos_x:double;  // position of object in world
    obj_pos_y:double;  // (X,Y,Z)
    obj_pos_z:double;
    obj_rot_x:double;  // rotation values around object-axis
    obj_rot_y:double;  // rotation-order: y,x,z
    obj_rot_z:double;  // axisses: y..up, x..right, z..to you
    obj_go_x:double;   // direction the robot looks
    obj_go_y:double;   // (is equal to the z-axis)
    obj_go_z:double;
    obj_name:PChar;    // name of the robot
  end;

  // a pointer to this record
  Peasybot_object_info=^easybot_object_info;

  // a pointer to a PChar-String ( in C char** )
  PPChar=^PChar;

  Easybot_Interfaces=class; // forward declaration

  // the interface
  DLL_Interface_Easybot=class
  public
    // name of your robot-navigation-module
    function GetName():PChar;
             virtual;cdecl;abstract;

    // version
    function GetVersion():PChar;
             virtual;cdecl;abstract;
```

```
// you
function GetAuthor():PChar;
         virtual;cdecl;abstract;

// get a description about your robot-navigation-module
function GetDescription():PChar;
         virtual;cdecl;abstract;

// get info about other interfaces in use
procedure SetEasybotInfo(p:Easybot_Interfaces);
          virtual;cdecl;abstract;

// get info about the sensor-rays
procedure SetRayInfo(p: Aeasybot_ray_info);
          virtual;cdecl;abstract;

// get info about the robot
procedure SetObjectInfo(p:Peasybot_object_info);
          virtual;cdecl;abstract;

// get messages from other robots
procedure SetMessageFrom(pfrom_robot,pmessage:PChar);
          virtual;cdecl;abstract;

// send messages to other robots
//(return 1: valid message, 0: no more messages)
function GetMessageFor(ppto_robot,
                       ppmessage:PPChar):integer;
         virtual;cdecl;abstract;

// next navigation step
procedure Navigate();
          virtual;cdecl;abstract;

// relative move in world-coordinates
function GetMoveX():double;
         virtual;cdecl;abstract;
function GetMoveY():double;
         virtual;cdecl;abstract;
function GetMoveZ():double;
         virtual;cdecl;abstract;

// rotation around axis in degrees
function GetRotationX():double;
         virtual;cdecl;abstract;
```

```
    function GetRotationY():double;
             virtual;cdecl;abstract;
    function GetRotationZ():double;
             virtual;cdecl;abstract;

    // open a dialog for options and status-info
    procedure OpenDialog();
              virtual;cdecl;abstract;

    // signals start(1) or stopp(0) of the simulator
    procedure SetStartStop(onoff:integer);
              virtual;cdecl;abstract;

    // interface is no longer needed, it can be destroyed
    procedure DestroyInterface();
              virtual;cdecl;abstract;
  end;

  Easybot_Interfaces=class
  public
    function GetFirstInterface():DLL_Interface_Easybot;
             virtual;cdecl;abstract;
    function GetNextInterface():DLL_Interface_Easybot;
             virtual;cdecl;abstract;
    function GetNumberOfInterfaces():longint;
             virtual;cdecl;abstract;

    function GetNumberOfRobots():longint;
             virtual;cdecl;abstract;
    function GetFirstRobotInfo():Peasybot_object_info;
             virtual;cdecl;abstract;
    function GetNextRobotInfo():Peasybot_object_info;
             virtual;cdecl;abstract;
  end;

implementation

end.
```

As you have seen it is a straight forward translation of the corresponding C header files.

2.5.2 A Basic Module

Now as we have got the interface definition, let us make a first try to create an extension module.

The Delphi Unit

We will create an own class which uses the interface as basis class, so we have to implement all these abstract functions of the interface. We will use the simplest possible implementation, which makes no important things at all. The only parts which are important here are the vectors *m_fmove* and *m_frot*. Both vectors are used as results for the simulator, so any child class has only to set these vectors and not to reimplement all these Get-functions.

We also save the references for the ray_info, object_info and interfaces in own variables, which can be used by child classes. (Rays is an Array, pObject is a pointer, Interfaces is a object reference).

Let us have look at the code (file: UEasybot_Basis.pas).

```
unit UEasybot_Basis;

interface

uses
  UDLL_Interface_Easybot;

type
  Vector=array[0..2] of double;

  Easybot_Basis=class(DLL_Interface_easybot)
  protected

    m_fmove,m_frot:Vector;
    Rays: Aeasybot_ray_info;
    pObject:Peasybot_object_info;
    Interfaces:Easybot_Interfaces;

  public

    constructor Create();

    function GetName():PChar;override;
    function GetVersion():PChar;override;
    function GetAuthor():PChar;override;
    function GetDescription():PChar;override;
```

```
    procedure SetEasybotInfo(p:Easybot_Interfaces);override;
    procedure SetRayInfo(p: Aeasybot_ray_info);override;
    procedure SetObjectInfo(p:Peasybot_object_info);override;

    procedure SetMessageFrom(pfrom_robot,
                             pmessage:PChar);override;
    function GetMessageFor(ppto_robot,
                           ppmessage:PPChar):integer;override;

    procedure Navigate();override;

    function GetMoveX():double;override;
    function GetMoveY():double;override;
    function GetMoveZ():double;override;
    function GetRotationX():double;override;
    function GetRotationY():double;override;
    function GetRotationZ():double;override;

    procedure OpenDialog();override;

    procedure SetStartStop(onoff:integer);override;

    procedure DestroyInterface();override;
  end;

implementation

constructor Easybot_Basis.Create();
begin
  m_fmove[0]:=0;
  m_fmove[1]:=0;
  m_fmove[2]:=0;
  m_frot[0]:=0;
  m_frot[1]:=0;
  m_frot[2]:=0;
end;

function Easybot_Basis.GetName():PChar;
begin Result:='Easybot Basis module'; end;

function Easybot_Basis.GetVersion():PChar;
begin Result:='1.0'; end;

function Easybot_Basis.GetAuthor():PChar;
begin Result:='René Liebscher'; end;
```

```
function Easybot_Basis.GetDescription():PChar;
begin Result:='Delphi Easybot Basis module'; end;

procedure Easybot_Basis.SetEasybotInfo(p:Easybot_Interfaces);
begin Interfaces:=p; end;

procedure Easybot_Basis.SetRayInfo(p: Aeasybot_ray_info);
begin Rays:=p; end;

procedure Easybot_Basis.SetObjectInfo(p:Peasybot_object_info);
begin pObject:=p; end;


procedure Easybot_Basis.SetMessageFrom(pfrom_robot,
                                       pmessage:PChar);
begin end;

function Easybot_Basis.GetMessageFor(ppto_robot,
                                     ppmessage:PPChar):integer;
begin result:=0; end;


procedure Easybot_Basis.Navigate();
begin end;


function Easybot_Basis.GetMoveX():double;
begin Result:=m_fmove[0]; end;

function Easybot_Basis.GetMoveY():double;
begin Result:=m_fmove[1]; end;

function Easybot_Basis.GetMoveZ():double;
begin Result:=m_fmove[2]; end;

function Easybot_Basis.GetRotationX():double;
begin Result:=m_frot[0]; end;

function Easybot_Basis.GetRotationY():double;
begin Result:=m_frot[1]; end;

function Easybot_Basis.GetRotationZ():double;
begin Result:=m_frot[2]; end;
```

```
procedure Easybot_Basis.OpenDialog();
begin end;

procedure Easybot_Basis.SetStartStop(onoff:integer);
begin end;

procedure Easybot_Basis.DestroyInterface();
begin
     self.Free;
end;

end.
```

As you could see there is neither a navigation algoritm built in nor any other specialisation as message transfer or dialog functions. This can be done by child classes.

The Project File

Now we have to create a appropriate project file to get a Windows-DLL.

To achieve this you have to change the module type from program to library (see first line of code) and you have to specify which function you want to export. We only need to export here a function called *GetEasybotInterface*. In this function you only need to create an instance of your navigation object and return a reference to it (in C++ it would be a pointer).

(file: delphi_basis.dpr)

```
library delphi_basis;

uses
  UDLL_Interface_Easybot in 'UDLL_Interface_Easybot.pas',
  UEasybot_Basis in 'UEasybot_Basis.pas';

function GetEasybotInterface():DLL_Interface_Easybot;cdecl;
var
   p:Easybot_Basis;
begin
        p:=Easybot_Basis.Create();
        Result:=(p as DLL_Interface_Easybot);
end;

exports
       GetEasybotInterface;

begin
end.
```

Now we have created our first DLL (do not forget to compile). Let's test it.

After you have load it, you should see name and description of our module in the dialog. Because we have not built-in any navigation the robot should not move.

Other Important Code Pieces

Message Transfer

You can send message from one robot to an other. To use this feature you have to override the following functions as follows:

```
procedure Easybot_Basis.SetMessageFrom(pfrom_robot,
                                       pmessage:PChar);
begin
  DoSomethingWithString(string(pfrom_robot));
  DoSomethingWithString(string(pmessage));
end;

function Easybot_Basis.GetMessageFor(ppto_robot,
                                ppmessage:PPChar):integer;
const
  to_robot='Khepera_1';
  hello='hello';
begin
  if MessageToSend() then
  begin
    ppto_robot^:=PChar(to_robot);
    ppmessage^:=PChar(hello);
    result:=1;
  end
  else result:=0;
end;
```

It is very important that if you send messages (GetMessageFor) the memory area where your message is in, is valid till the next call of GetMessageFor. A good idea would be to create two member variables for it, which will be set right before you send a message and stay unchanged till you send a next message.

OpenDialog

To use a graphical interface for your module you can create it as normal Delphi form. And open it in the function OpenDialog.

```
procedure Easybot_Basis.OpenDialog();
begin
  Dialog.Create();
end;
```

However it would be more appropriate to store a reference to the dialog in a member variable, so you can close and destroy it, if your module is going to be end. And you do not open every time a new dialog, instead you could make the existing one visible.

The EasyBot_Interfaces Object

At the moment this part of the interface is not working because of differences in built-in interface and ours, but because we need a vtable, it is probably that the built-in interface is going to change to abstract virtual definitions in C++ too.

2.5.3 Movement Controlled by Wheel Speeds

After this simple module, let us extend it to functionality which is essential, if we want to use it for simulating Kheperas. Kheperas are controlled by the speed of their wheels, but the simulator wants a movement vector in world coordinates, so we have to do some computationen to get it.

How the Computation Works.

The principle is fairly easy and should be known to everyone, who ever used odometry with the khepera. It is possible to calculate from the wheel speeds (or better their proportions) and the distance between the wheels a radius which describes the movement about a certain point. The angle can be calculated by the way on this circle, which is dependent on the time step and the speed of the robot on this circle (the average of the wheel speeds).

So it should be easy to understand if have a look at the code.

An other aspect is that the simulator uses 3D coordinates and a robot could move up a rise, this would mean that we have to turn all robot local coordinates into world coordinates. Again such a computation is a standard algorithm, which consists of multiplications of rotation matrices and vectors.

And the Code

In the Navigate function is the computation done. To work with matrices and vectors, we need some additional functions too.

(file: UEasybot_Khepera)

```
unit UEasybot_Khepera;

interface

uses
  UEasybot_Basis;

const
  deltaX=0.455; // distance between wheels in units
                // ( 1 unit= 0.1 meter )

  vmax=10.0; // maximum velocity in units per second

type
  Matrix=array[0..2] of array[0..2] of double;

  Easybot_Khepera=class(Easybot_Basis)
  public
    deltaT:double; // simulation time step in seconds
    v:array[0..1]of double;
    // -1..1 velocity of wheel relative to
    //       maximum velocity

    constructor Create();

    function GetName():PChar;override;
    function GetVersion():PChar;override;
    function GetAuthor():PChar;override;
    function GetDescription():PChar;override;

    procedure Navigate();override;

  end;

implementation

uses
  SysUtils, // Exception
  Math;
```

```
// using MessageDlg from unit Dialogs would result
// in 250 kB more in file size
function MessageBox(HWnd: Integer;
                    Text, Caption: PChar;
                    Flags: Integer):Integer;
         stdcall;
         external 'user32.dll' name 'MessageBoxA';


procedure setVector(var v:Vector;
                    values:array of double);
var i:integer;
begin
  for i:=0 to high(v) do
    if i<=high(values) then v[i]:=values[i]
                       else v[i]:=0.0;
end;

procedure setMatrix(var v:Matrix;
                    values:array of double);
var i,j,k:integer;
begin
  k:=0;
  for i:=0 to high(v) do
    for j:=0 to high(v[i]) do
    begin
      if k<=high(values) then v[i][j]:=values[k]
                         else v[i][j]:=0.0;
      inc(k);
    end;
end;

procedure MultMatMat(var m1,m2,m3:Matrix);
var
  i,j,k:integer;
  sum:double;
begin
  for i:=0 to 2 do
    for j:=0 to 2 do
    begin
      sum:=0.0;
      for k:=0 to 2 do
        sum:=sum+m1[i][k]*m2[k][j];
      m3[i][j]:=sum;
    end;
```

```
end;

procedure MultMatVec(var m1:Matrix;
                     var v1,v2:Vector);
var
  i,j:integer;
  sum:double;
begin
  for i:=0 to 2 do
  begin
    sum:=0.0;
    for j:=0 to 2 do
      sum:=sum+m1[i][j]*v1[j];
    v2[i]:=sum;
  end;
end;


constructor Easybot_Khepera.Create();
begin
  deltaT:=0.05; // 50ms time step
  v[0]:=0.1;
  v[1]:=0.2;
end;

function Easybot_Khepera.GetName():PChar;
begin Result:='Easybot Khepera basis module'; end;

function Easybot_Khepera.GetVersion():PChar;
begin Result:='1.0'; end;

function Easybot_Khepera.GetAuthor():PChar;
begin Result:='René Liebscher'; end;

function Easybot_Khepera.GetDescription():PChar;
begin Result:='Delphi Easybot Khepera basis module'; end;


procedure Easybot_Khepera.Navigate();
var
  // movement in locale coordinates
  locale_vector:Vector;

  // matrices for calculations
  rotY,rotX,rotZ,result1,result2:Matrix;
```

```
  // radius and turning angle
  r,alpha:double;

begin

  try // we dont't want to crash if something goes wrong

    // matrices for rotations
    setMatrix(rotY,
    [
      cos(DegToRad(pObject^.obj_rot_y)),
      0,
      sin(DegToRad(pObject^.obj_rot_y)),

      0,1,0,

      -sin(DegToRad(pObject^.obj_rot_y)),
      0,
      cos(DegToRad(pObject^.obj_rot_y))
    ]);
    setMatrix(rotX,
    [
      1,0,0,

      0,
      cos(DegToRad(pObject^.obj_rot_x)),
      -sin(DegToRad(pObject^.obj_rot_x)),

      0,
      sin(DegToRad(pObject^.obj_rot_x)),
      cos(DegToRad(pObject^.obj_rot_x))
    ]);
    setMatrix(rotZ,
    [
      cos(DegToRad(pObject^.obj_rot_z)),
      -sin(DegToRad(pObject^.obj_rot_z)),
      0,

      sin(DegToRad(pObject^.obj_rot_z)),
      cos(DegToRad(pObject^.obj_rot_z)),
      0,

      0,0,1
    ]);

    // no movement as default
```

```
    setVector(locale_vector,[0.0,0.0,0.0]);

    // no turn as default
    setVector(m_frot,[0.0,0.0,0.0]);

    // movement in locale coordinates
    if v[0]=v[1] then
    begin // straight movement
      // forward distance (Z)
      locale_vector[2]:=v[0]*vmax*deltaT;
      m_frot[1]:=0; // no turn
    end
    else
    begin
      r:=(v[0]+v[1])/(2*(v[1]-v[0]))*deltaX;
      alpha:=deltaT*(v[1]-v[0])*vmax/deltaX;
      // sideward distance (X)
      locale_vector[0]:=((1.0-cos(alpha))*r);
      // forward distance (Y)
      locale_vector[2]:=(sin(alpha)*r);
      // turn about own Y-axis
      m_frot[1]:=RadToDeg(alpha);
    end;

    // compute world coordinates from locale coordinates
    MultMatMat(rotY,rotX,result1);
    MultMatMat(result1,rotZ,result2);
    MultMatVec(result2,locale_vector,m_fmove);

  except
    { ignore errors e.g. EZeroDivide,EOverflow,EMathError }
    on E: Exception do MessageBox(0,PChar(E.Message),NIL,16);
  end;
end;

end.
```

Now we control our robot by setting the appropriate values in the array *v*. In the constructor are some example values set, so if you try this module, the robot should turn left.

By changing the value of *deltaT* you can control the length of the simulated time span (only concerning the moved distance).

2.5.4 A Simple Example

On this basis we can now build up and start create control modules without great efforts for low level computations.

What you still have to do, is to calculate sensor values from the length field you find in the Rays array and to translate wheel control commands in the -1 to 1 area of the basis module. Also you have to be aware possible collisions with other objects (walls,...), because the simulator ignores any collisions.

And now to complete, a short example which shows what is left to do to get navigation module.

Its Task

This example does not implement a serious control algorithm, it simply steers the robot using two sensors to avoid walls.

And the Complete Code

As you can see now, it is much less to program as if you would have to implement the complete interface for yourself.

(file: UEasybot_Khepera_Ex.pas)

```
unit UEasybot_Khepera_Ex;

interface

uses
    UEasybot_Khepera;

type
    Easybot_Khepera_Ex=class(Easybot_Khepera)
    public
        constructor Create();
        procedure Navigate();override;
    end;

implementation

uses
  Math;

constructor Easybot_Khepera_Ex.Create();
begin
```

```
  // no initialization necessary
end;

procedure Easybot_Khepera_Ex.Navigate();
begin
  // simple algorithm, which uses only 2 sensors
  // (front left, front right) to avoid walls
  v[0]:=min(Rays[4].ray_length,1)*0.1;
  v[1]:=min(Rays[1].ray_length,1)*0.1;
  inherited Navigate();
end;

end.
```

(file: delphi_khepera_ex.dpr)

```
library delphi_khepera_ex;

uses
  UDLL_Interface_Easybot in 'UDLL_Interface_Easybot.pas',
  UEasybot_Khepera_Ex in 'UEasybot_Khepera_ex.pas';

function GetEasybotInterface():DLL_Interface_Easybot;cdecl;
var
   p:Easybot_Khepera_Ex;
begin
        p:=Easybot_Khepera_Ex.Create();
        Result:=(p as DLL_Interface_Easybot);
end;

exports
       GetEasybotInterface;

begin
end.
```

2.5.5 Other Pascal Compilers

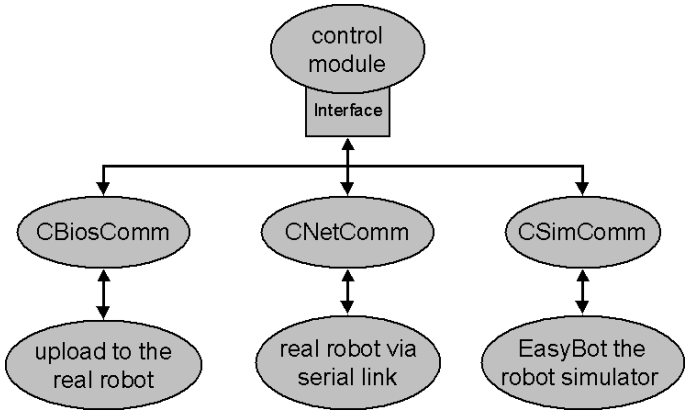
There is a free Pascal Compiler (www.freepascal.org), which is also able to create Windows-DLL's. This compiler should also be able to use these modules, because it has a Delphi compatibility mode (but it was not tested yet). However, a drawback is, it does not offer facilities like Delphi concerning the building of graphical interfaces.

Chapter 3

Interface Design

3.1 Concepts

Our main intention in the interface design was, to build up a common interface for all kinds of using. That means we intended to create an interface which allows, the programmer of a robot control unit, to generate one module that can be either used within the robot simulator, control the real robot via the serial link, or used on the real robot after compiling and downloading on it.



3.2 Realization

To realize such an interface we had to generate a common structure for all three cases.

When we built up the structure, we did not want to decrease the variety of abilities the Khepera robot has. Otherwise we had to cope with the differences the robot has in using it via serial port, and by downloading a compiled programm to the robot. These problems we solved building up the interface in a way that provides all possibilities of the serial link communication. We ignored the additional capabilities given by the bios (when downloading a program). From our point of view, these capabilities were not necessary for a common usage of the robot, as they can modify the scheduling and internal behaviour of it. The general scheduling and behaviour of the robot should be OK in the most cases of usage.

3.2.1 The Interface

The header file contains the general interface of the serial link, as they are desribed in the Khepera User Manual [4].

```
#ifndef _NET_COMMAND_H_
#define _NET_COMMAND_H_

#include "commport.h"

typedef struct _sensors_{
   long left_90;
   long left_45;
   long left_10;
   long right_10;
   long right_45;
   long right_90;
   long back_right;
   long back_left;
} sensors;


class CNetCommand
{
   public:
      CNetCommand();
      ~CNetCommand();
      void SetCom(CCommPort *com);
      bool Configure(long kp,long ki,long kd);
      bool Read_software_version(float* bios_version,
                  float* protocol_version);
```

```
      bool Set_a_position_to_be_reached(long left,
                                        long right);
      bool Set_speed(long left,long right);
      bool Read_speed(long* left,long* right);
      bool Configure_the_position_PID_controller(long kp,
                      long ki, long kd);
      bool Set_position_to_the_position_counter(long left,
                                                long right);
      bool Read_position(long* left,long* right);
      bool Read_AD_input(long channel_no,long* analog_value);
      bool Configure_the_speed_profile_controller
         (long max_speed_left, long acc_left,
          long max_speed_right, long acc_right);
      bool Read_the_status_of_the_motion_controller(
         long* T_left, long* M_left, long * E_left,
         long* T_right, long* M_right, long* E_right);
      bool Change_LED_state(long led_no,long action_no);
      bool Read_proximity_sensors(sensors* sensor);
      bool Read_ambient_light_sensors(sensors* sensor);
      bool Set_PWM(long left, long right);
      bool Send_a_message_to_an_extension_turret(
         long turret_ID, char* command,char* response);
      bool Read_a_byte_on_the_extension_bus(long rel_addr,
                                            long* data);
      bool Write_a_byte_on_the_extension_bus(long data,
                                             long rel_addr);
   private:
      CCommPort   *com;
      char     buf[1024];
      bool Read_string(char *retstr);
};
#endif
```

3.2.2 Simulation "sim_command"

EasyBot, the robot Simulator, is controlled by a DLL. To be able to use our interface we implemented `sim_command`. It contains the conversion between EasyBot's DLL-internal structures (see Chapter 2.3) and our interface.

This implementation is responsible for the complete behaviour of the robot simulation. That means, we set the speed for each wheel, when control function is triggered. `sim_command` has to compute the position and orientation of the robot. It could be extended to simulate a real environment, including collision detection, friction, dynamics ...

As EasyBot does, it works only in a Windows environment.

3.2.3 Serial Communication "net_command"

The serial link protocol is based on plain text strings. Commands are represented by a single character, followed by their parameters. Results are sent back as strings as well. The implementation inside here converts our interface to these commands. The communication works synchronously, that means after sending a command to the robot, the interface waits for an answer.

3.2.4 Download to Khepera "bios_command"

To use Khepera internal system calls, you can compile your own *C*-controls to run directly on the robot. Our intention of this implementation, is that you do not use the robot-internal commands, but our net_command interface. This implementation converts these commands to the Khepera system calls. The final version needs to be compiled (see Chapter 5.1) and downloaded as described in Chapter 6.1.

3.3 Usage

3.3.1 Implementation Issues

The user of our interface has to implement three functions:

- `void ConstructData(CNetCommand *nc);`
- `void ControlFunction(CNetCommand *nc);`
- `void DestructData(CNetCommand *nc);`

The description of these functions can be found later in this manual (see 4.1).

When the control routines for the robot are complete they just need to be compiled and linked against the library depending on the special case of usage. This allows to test a control module using the simulator before trying it on the real robot.

3.3.2 Generating Code

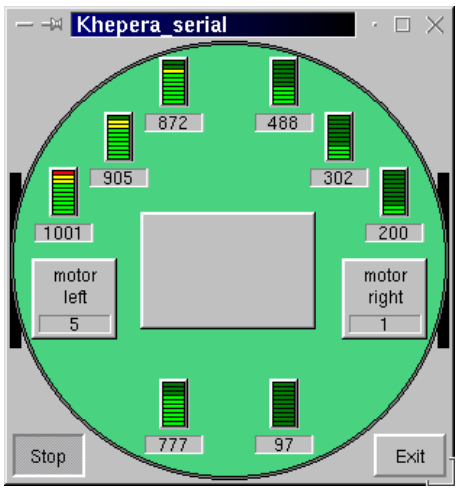
Windows based "sim_command"

To connect our controll to EasyBot we need a DLL. This DLL is prepared with a library that can be bound to your control-module. Therefore you use our Makefile by calling

`nmake control=`*control-name*

in the `DLL_SimCom`-directory. The contol-name has to be given with its extension (usually `.cpp`). The output file can be found in the `Release`-subdirectory named `DLL_SimCom.dll`. The Makefile uses the Compiler which is part of the Microsoft Developer Studio.

Linux based “bios_command” and “net_command”



After extracting the Khepera interface (Khepera.tgz) you just copy your C++-control-file into the main directory of it (directory Khepera).

Assuming your file has the name `mycontrol.cpp` you just call

```
make mycontrol
```

and make automagically creates two executable files (`serial_mycontrol` and `qserial_mycontrol`). The `serial_mycontrol` simply connects the Khepera with the control via its serial connection. The `qserial_mycontrol` works similary to `serial_mycontrol`. Additionally it shows the state of the robot using the Qt-GUI-toolkit. With this control you can see sensor-values and wheel-speeds. Here you can start and stop the robot. When you stop the robot, the wheel speeds are set to zero. The sensor values are still displayed. The robot is stopped as well when you leave the program. The Qt-frontend of your control depends on images stored in the `img`-subdirectory of its destination path. Thats why you need to copy this directory as well if you want to move your compiled program.

In later releases it will be able to also create executables which can be downloaded into the robots RAM (“bios_command”).

Chapter 4

Examples

4.1 A Simple Control Module

In this section we try to describe how to program a control module that matches our interface for all three usages of the Khepera (simulator, serial link mode and autonomous). The task of the controller is to move without hitting a wall in first place, and to demonstrate basic ideas of programming such a module.

Firstly you have to include the `control.h` header-file that contains the declaration of the variables and functions to implement for a control module.

Here the source code of our small example:

```
#include "control.h"

// these values (PortNumber and Speed) have to be defined
// here, even if you do not want to change/use them
int PortNumber=1;
int PortSpeed=9600;

void ConstructData(CNetCommand *nc)
{
   // generate your data structure here and pass
   // a pointer to the SetData function;

   int * status;

   status = new int;
   *status=0;
   nc->SetData(status);
   nc->Configure_the_speed_profile_controller(60,10,60,10);
```

```
}

//this function is running in an endless loop

void ControlFunction(CNetCommand *nc)
{
   long l_speed,r_speed;
   long max_speed=50;
   sensors sens;
   int * status;

//      access the data structure
   status = (int*) nc->GetData();

//  just to (academically) use status
   if (*status==0) *status=1;

   nc->Read_proximity_sensors(&sens);

   l_speed=r_speed=0;
   l_speed=(9*(1024-sens.right_10)
          +7*(1024-sens.right_45))*max_speed/16400;
   r_speed=(9*(1024-sens.left_10)
          +7*(1024-sens.left_45))*max_speed/16400;
   nc->Set_speed(l_speed,r_speed);
}

void DestructData(CNetCommand *nc)
{
   int * status;

   status = (int*) nc->GetData();
   nc->SetData(NULL);
   delete status;
}
```

Each function contains a parameter of the `CNetCommand`-type. This is the class to communicate with the robot. There are all commands available you can use for the serial link communication. These commands are ported for the usage in the other modes. This commands are described in the Khepera User Manual (App. A) [4]. The exact definition of the interface can be obtained from the `net_command.h` header-file.

For your own purposes you probably need your own variables. Caused by the internal usage of multithreading, these variables can not be globally defined. Inside of the `CNetCommand`-class exists a possibility to store user data. To use this you need to

hand over *one* structure containing all your permanent variables. Local variables can be used as usual.

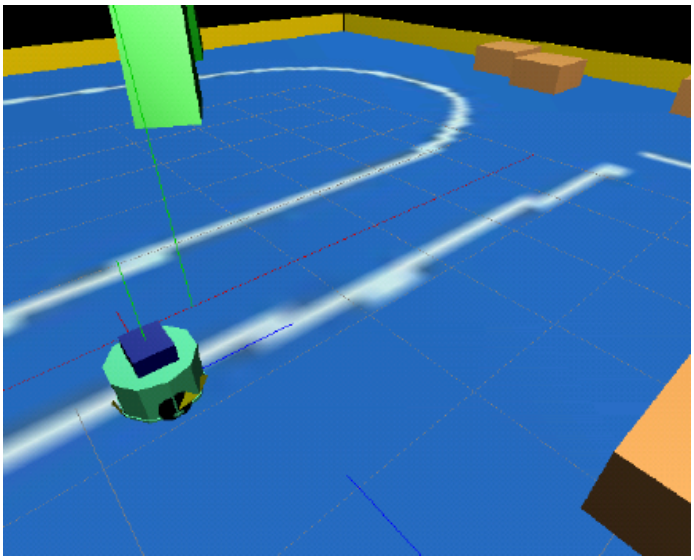
For a control module you need to implement three functions:

`void ConstructData(CNetCommand *nc)`
: Inside this function yous have to create (`malloc(...)` or `new(...)`) and initialize (if necessary) your data structure. After generation you call:
`nc->SetData((void*)` *pStructure*`);`

`void ControlFunction(CNetCommand *nc)`
: This function is running in a loop, triggered by external events. It contains one step of control. When acessing the function you need to call
pStructure `= (`*Structure-type*`*) nc->GetData();`
to get back your data. Now you can manipulate the robot and your data, before the function has to return.

`void DestructData(CNetCommand *nc)`
: Finally this function needs to destroy your data structure using `free(...)` or `delete(...)`. Therefore you need to get your data structure as in the `ControlFunction`, and write a `NULL`-pointer back as in `ConstructData`.

4.2 Line tracking



4.2.1 Task

An interesting task for a robot could be to find a target. There is a white line on the ground to help the robot. The robot has to follow this line. There are real systems using this method. For example there are autonomous transport robots in car-manufacturing.

4.2.2 Hardware

There are several extension turrets for the Khepera robot. One of them, the K213-extension, is a 1-line-camera. It scans 64 pixels on the ground in front of the robot, in a distance of approximately 15 cm. The camera returns grey-scale values.

Further information can be found in the K213-manual [6].

4.2.3 Algorithm

The algorithm analyzes the whole scan-line to find the lightest position on it. This is considered as the centre of the line. The aim is to keep the line in the middle of the scan-line. The more the line moves out of the centre of view, the more the robot has to rotate.

The algorithm analyzes the brightness at certain points in the scan-line. If there are points on the left side, which are brighter than a fixed level, they cause a rotation to the

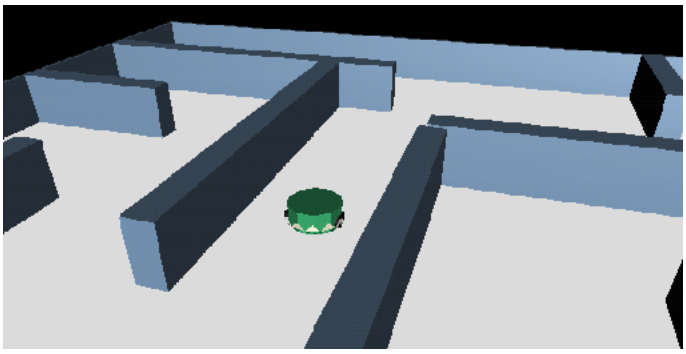
left. The more a scanned point varies from the centre, the higher is the angle to rotate. Left (negative) and right (positive) angles are added, and multiple lines can result in zero.

With a special option (“rechtwinklige Ecken ausfahren”) this line-algorithm prefers straight lines. This means if there is a white point in the centre of the scan-line, it ignores all other input. When the centre point gets lost, the robot keeps on walking for a while. After a timeout it is assumed, that the robot has reached the end of the line. Now it rotates to find a new line. This is done by looking 90 degrees to the left, then to the right. If it did not find anything, the robot continues walking straight forward.

4.2.4 Comments

The source code is to be found in the section that describes how to create a control-DLL (see 2.3). In addition this control contains code to avoid collisions.

4.3 Orientation in a Maze



4.3.1 Task

One of the most important capacities of a robot should be that it can find its way out of a foreign area and return to the base. Some of these areas are organized like a maze, for instance if the robot explores a cave or similar. Also a road system is sometimes like a maze. So we decided that we use a maze to develop basic orientation routines.

4.3.2 Theory

To develop orientation routines for a robot we have to take into consideration that the robot only has its eight distance sensors. It doesn't have absolute coordinates. Calculating relative coordinates using the wheel movement is very difficult and also not precise because you don't have information about altitude and friction.

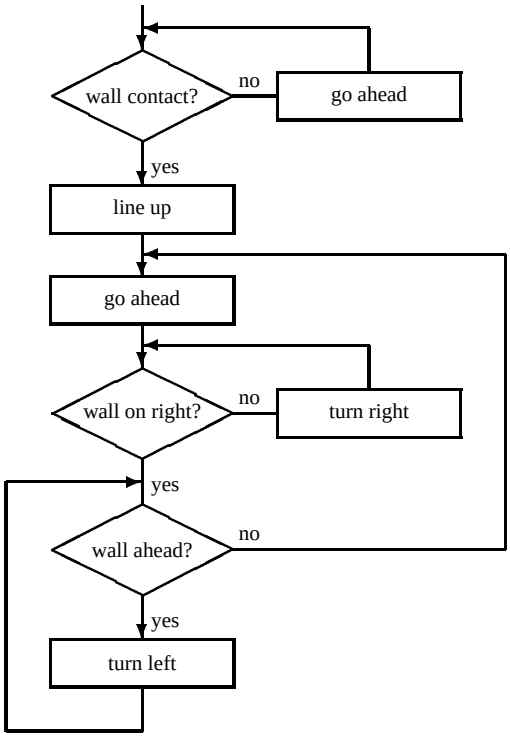
Our idea was that if you have a maze without internal loops you find a way out by following a wall. Using a maze with loops like blocks in a road system is nearly impossible without coordinates.

4.3.3 Traditional Program

The first step for the robot is to look for a wall. If its sensor doesn't see one, the robot starts moving straight on till it sees one and turns, until this wall is on its right side. Then it trys to follow this wall. We decided that the robot should use the right wall. If the sensors on the right don't see the wall any more the robot turns right. If there is a wall on the right side but none in front of the robot it moves straight ahead. If there are walls on the right side and in front of the robot the robt turns left till it finds a way straight on.

Of course, it is also possible to go along a left wall but we had to decide for one of these possibilities.

There's a sample flow diagram for our algorithm:



4.3.4 Neural Network

While running the algorithm in the simulator we recorded the input data and the calculated output to the wheels into a file and tried to teach several neural networks with this data in the Stuttgart neural network simulator [7]. We got the best results with a network with one input layer with 8 neurons (for the eight sensors), a hidden layer with 3 neurons and two output neurons for the wheels. The best learning method was rprop. With snns2c it's easy to make a C-file from the network and use it in the robot simulator.

In a second step we pruned the network. About 40% of the connections were deleted during the pruning process, and also one of the sensors wasn't used anymore. This network was much faster than the old one and ran as good as the unpruned network.

4.3.5 Possible Improvements

At the moment our algorithm and also the network made the robots movement very crude. A smoother movement would be better. The robot also has problem recognising small interferences, and knocks itself on sharp corners.

Another improvement could be the usage of genetic algorithms and train it in several networks. There it would be possible to teach the robot to avoid loops and find a faster way out of the maze.

4.4 Genetic Algorithms

4.4.1 "Surviving"

Aim was, to use a genetic algorithm to control robots. What's to do? Certainly it has to survive. Therefore we need a genetic algorithm for surviving, more exactly for not-driving-through-walls. How can a robot control work with genetic algorithm? Not at all – except we use a neural network as well.

The neural network uses the sensor values and actual speeds as input and computes the new wheel speeds. The neural network works without hidden layers, to keep the example as simple as possible. To get some genetic stuff in there, the weights of the network are set by genetic algorithms!

Though we have 10 input and 2 output neurons. This gives us 20 weights to adjust. Therefore each individual of the genetic algorithm has 20 characteristics (genes).

Now we create a uniform basis for all individuals. They start all in the same place with the same view. Each population consists of 20 individuals. Let's start them and look how long they survive. Surviving is defined by not driving against a wall. At the beginning, the weights are initialized randomly. This causes that all robots will probably turn around their axis, since both motors are not operating equally. To avoid them turning forever, a maximal life span has to be determined (here 2000). If they drive against a wall before, of course their lifetime is over.

Since the 20 individuals would obstruct themselves mutually, we let them learn one by one.

Let the first generation "rotate", or whatever it does. What happens? We probably have 20 robots, which had to be stopped all from the outside (maximum lifetime reached). How can we decide, which were stronger or weaker individuals? Certainly we do not take just the lifetime, but also the way they did as a benchmark for the strength (fitness) of a genome. We used this formula:

$$\text{fitness} = \text{bee-line} \cdot \text{way} \cdot \left(\frac{\text{lifetime}}{2000} + 1 \right)$$

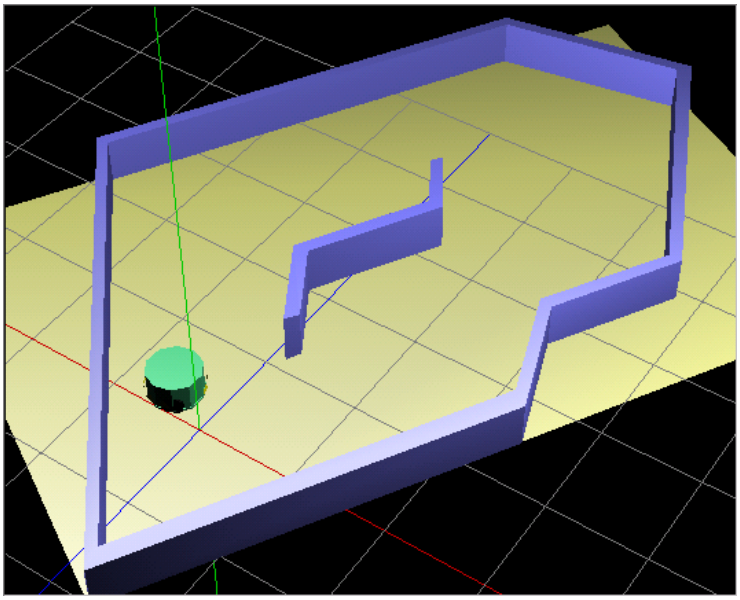
Bee-line is intuitive. The way is the accumulated distance the robot moved. If it just rotates, it does not cover any distance. To avoid the lifetime carrying too much weight, we scaled it to a range between 1 and 2! This should give us a fair evaluation.

Let's go back to our population. The first generation was lively rotating and each robot covered thereby more or less distance. We determine the fitness of the individuals and select the best for reproduction. To give strong "parents" a chance, just the weaker half of the parent generation is replaced. The stronger may live on. That they do not make the same "mistakes" like in the passed generation, we mutate genome slightly. The production of the "children" takes place by the principle of ranking. All individuals are sorted according to their fitness. The chance to reproduce depends on their rank. The new individuals are generated by cross over followed by a mutation. In this way we get 10 new individuals and 10 from the previous generation with small modifications.

Now we start all over again. All may turn around, but the better they run straight forward and runaunaround the walls, the higher is their fitness and their chance to reproduce.

To test this theory and algorithm we use EasyBot. It offers the advantage that we do not have to reset by hand. The program does it. Now we can train the algorithm without a permanent control. Additionally the number of broken robots, which crashed against a wall, is substantially smaller.

We used the usual maze of Francesco Mondada to train:

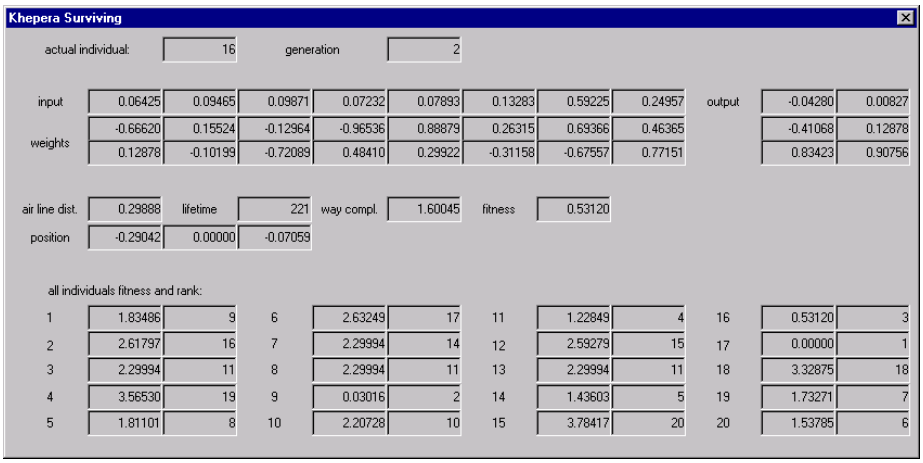


If the algorithm generates a robot that survives in the maze, it means this individual is a strong one. We suppose there is a strong robot, that runs around the center piece. In this case, it is possible that the robot returns to its starting point when their lifetime is over (timeout, 2000). This robot gets in trouble during the bee-line rating. In extreme cases this is even zero. Therefore its better to amend the formula a little:

$$\text{fitness} = (\text{bee-line} + 0.01) \cdot \text{way} \cdot \left(\frac{\text{lifetime}}{2000} + 1 \right)$$

By this modification even such robots have a chance not to get lost. Well, this problem represents not actually a large obstacle. In the stage, in which it generates a robot that orbits the maze, there is probably another one which does it as well. Additionally the probability is very small anyway.

4.4.2 Let's try



In the last chapter we said, all robots would turn around. That's not true. Most of them do, but surprisingly, in the first generation were some good individuals too. The algorithm works, but in the first generation, most of robots drove backwards. It took some generations, to get all robots to drive forward. Another problem was, after 20 generations there were 20 strong individuals, which all drove into one corner. There the life-time of them expired. This behaviour caused all new individuals to drive to this corner as well. To avoid this, we should change the algorithm a little bit. For example we could extend the life time, drive a little bit faster or mutate more of the genes.

However, we were happy, that it worked and that the robots really "ran" around the walls. Currently we have a basis to experiment with robots and walls and surviving or whatever you want.

4.4.3 Summary

Finally we have a population, which goes on in the evolution. It is controlled by a genetic algorithm, which generates a quite stable and evenly strong population after a certain number of generations. Each individual is controlled by its genome. This is used as weights for a neural network, where the 8 sensors (quasi eyes) of the robot and its previous speeds are used as input, and it supplies directly the new speeds.

4.5 Genetic Algorithm for Training a Neural Network

Important: This module was developed to play around with the simulator and genetic algorithms, do not take it too seriously.

Here we present an example plugin which uses a genetic algorithm to train neural network. The network is a multi layer feedward network which has a hidden layer with three neurons. The inputs are the values of the proximity sensors and the current speed of the wheels (in this sense it is more a recurrent network). The output neurons control directly the wheels.

The genetic algorithm is used to adjust the weight of this network. They are stored as gray-coded fix-point numbers (-128.0, 127.0).

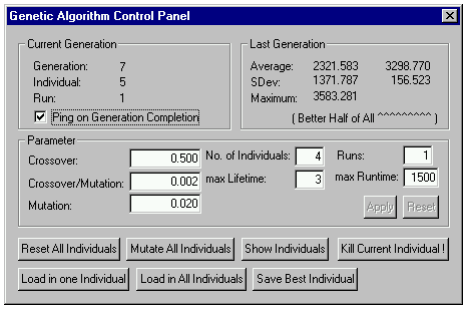
All individuals will be tested with the same start conditions (position 0,0,0) and the best will used as parents for the next generation (this is a kind of elitism). The new individuals will created by crossover each with each other and the mutated parent individuals. However, if the children are not better than their parents, the children will not survive their parents, but the parents can only survive a certain number of generations.

The fitness function is defined as the product of the length of the way, which the robot moved in forward direction, the maximum distance the robot reached from its origin and one plus its lifetime in steps divided by 1000.

$$fitness = way\_length \cdot max\_distance\_to\_origin \cdot (1 + \frac{lifetime}{1000})$$

If the robot comes to close to a wall (0.05 units), it dies. (This is considered as a crash into the wall, however you could also see the walls as a kind of electric fence, where robot gets a deadly electric shock. A real robot would stop at wall, still trying to move into the wall).

To use this controller you can open the following dialog:

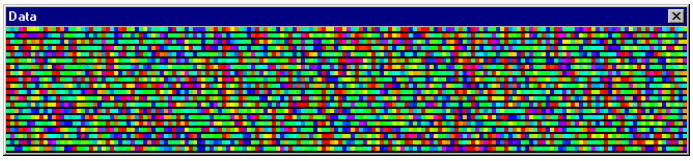


In the upper left corner you can see which individual is current active. At the right you can see some statistics about the last generation. In the middle part are the parameters for the creation procedure of new individuals as well as some entries which control the algorithm as whole, which means how many individuals survive as parents their

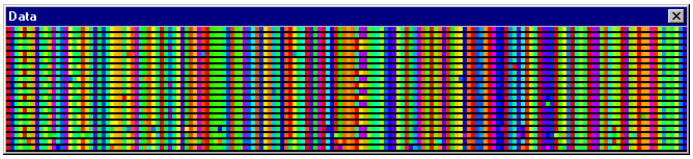
lifetime and the maximum runtime of each individual (the number means simulation steps).

To start the algorithm you have to start the simulator in the usual way. It is possible to speed up the algorithm by enter as step time 0 milliseconds in the main panel. (However, you get then no screen updates, and you should avoid everything which forces the program to redraw the 3D-image, because there is a serious synchronisation problem in this special case in the current LV3D version. This problem can destroy the internal geometry of the robot. So the best is, you make the main window so small that you cannot the 3D-image so it will also never get redraw events).

If you activate the button ‘Show Individuals’ you get a panel like this:



This panel shows the genes of each individual as colored strings. It is only intended to show, how individuals are similar or different. After some generation it shows something like the following picture, which means the individuals are now much more similar than they were at the start.



To try this example you should use the environments labyrinth*.lvs or ObstacleAvoidance.lvs. The latter one is modelled after an experiment which was originally carried out by Francesco Mondada and had also a genetic algorithm as research subject.

Chapter 5

Code Generation Using the Cross-Compiler

5.1 Khepera and the GCC-Compiler

The Khepera robot has the possibility to act in several modes:

serial the commands come via a serial link and are interpreted by the bios of the robot, the whole logic resides outside the robot in the computer it is connected to

download the program for the robot is crosscompiled and downloaded to the robots RAM where it executes – the robot controls itself

rom the program is crosscompiled and burned into an EPROM which is then plugged into the robot – this way the program “survives” resets

Khepera itself has an M68020 processor which can be targetted by gcc using the m68k processor model. The most common OS on that class of processors was SunOS version 2 and 3 — so it seems to be logical to use this model as target-host to configure the gcc as cross-compiler.

The K-Team deliveres a standard libgcc.a and some includes for the compiler to be able to compile C-Programs for it. All the C examples and our trials with C worked fine with this configuration. But we had problems with C++.

That far the Khepera Cross-Compiler manual [5] describes the process of creating the compiler quite well. So we leave it to you to follow their steps.

5.1.1 Triple trouble

Even if you compile gcc with the option `"LANGUAGES=C C++"` it won't correctly link sources which use certain C++ features (like classes with destructors and virtual methods). If you test the binary with sun2-nm you will detect three symbols with the type prefix "U" instead of "T" or "D". These symbols are `_terminate_Fv`, `___eh_pc` and `___throw`.

Just redefining these symbols enclosed in `extern "C" { }` didn't help anything: they have to do their job.

Sadly the Khepera Team – which developed this library delivered with the robot – left us alone with this problem. The answer to our e-mail is still outstanding. Although libgcc is part of a GNU GPL'ed program they don't seem to like the GPL paragraphs which state that you have to deliver sources with GPL'ed programs.

According to our explorations the Khepera libgcc.a contains these parts:

bios.o the one and only real Khepera object file, it provides the functions which control the Khepera hardware.

libgcc1.a the basic libgcc functions which satisfy all needs of a C Compiler and a C++ compiler which doesn't compile C++ sources.

libc parts of the libc like the symbols from string.h and so on.

There seem to be two ways of solving the problem of the missing symbols.

Completion of Kheperas libgcc: It should be possible to compile a libgcc.a for the new compiler (call `make libgcc.a` in the gcc source directory), then extract with ar the object files, which contain the needed symbols and include them to Khepera's libgcc.a. Other libraries "needed" by the linker could be replaced by a simple dummy DLL, which could be made this way:

```
echo >dummy.c
sun2-gcc -c dummy.c
sun2-ar rs dummy.a dummy.o
```

Build a new libgcc/libc: This is much more complicated. You would have to build a new libgcc, libc, libm and libstdc++ for Khepera and then extend the libgcc by bios.o which you may extract from the old Khepera libgcc. The tricky part is to drive the library makefiles wild enough to crosscompile instead of just compiling for the host system – I guess this requires some hacking inside the makefiles and some tricks with the compiler binaries.

Sadly we did not have time enough to go either of these ways.

Chapter 6

Tools

6.1 Khepera Upload Tools

These tools are designed to upload cross-compiled programs to the Khepera robot. These programs usually have the extension `.s28`. The Khepera has to be in download mode (see Khepera User Manual [4]) connected to a serial port. When the program is on the robot, it can be disconnected. The Khepera robot works autonomously.

6.1.1 KhUpload for LINUX

On the LINUX platform KhUpload is a command line tool. The file name, port number and speed of transmission are passed as parameters to the program.

```
KhUpload <port> <file> [<speed>]
```

<port> contains the file-descriptor of the Linux special file for the serial port. This is `/dev/ttyS0` for `COM1`, `/dev/ttyS1` for `COM2` ...

<file> stands for the cross compiled verion of your control `.s28` using `bios_command`.

<speed> can be used to set the speed for the transmission. The default speed is 38400 [bd]. Alternatively you can change to 9600 or 19200.

6.1.2 KhUpload for Windows NT4

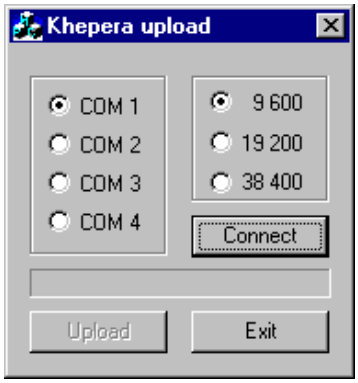


Figure 6.1: Main dialog of the KhUpload program.

To upload a program you first need to connect the serial port. Therefore you have to select portnumber and speed before clicking on the “Connect”–button. In the moment the program is connected to the serial port the “Upload”–button becomes enabled. If you use this button the file open dialog appears.

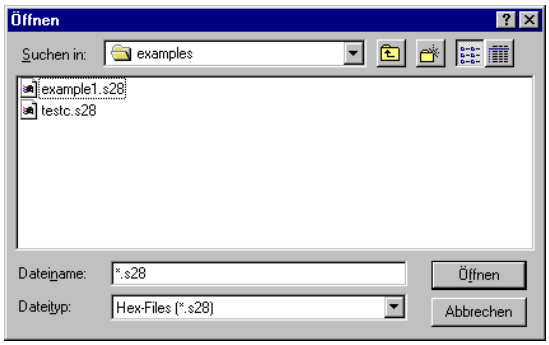


Figure 6.2: File Open Dialog of the “KhUpload” program

Inside the file open dialog you can select s28–files. If you select such a file the upload starts immediately. The progress of the upload is shown in the progress bar in the middle of the main dialog.

Chapter 7

Future Extensions

7.1 Interfaces

Up to now there is just the concept and the core functionality of the common interface completed. Due to the lack of time it was impossible to implement the whole functionality of `sim_command`.

As descibed within the compiler section the problems in cross-compiling need to be solved.

Another thing to do is to transfer all tools to both platforms (WindowsNT and Linux).

Finally it is probaly necessary to syncronize the behaviour of all three interfaces (`bios_command`, `net_command` and `sim_command`). Therefore it is probably necessary to fiddle around with time bases, speeds and other funny stuff.

7.2 Simulator

A physical model ("real world" with gravity and friction) in the LV3D_easybot-environment could be implemented. Then the simulated Khepera-robots could slip on the ground without their wheels turning. Then the wheels would slow down the motors according to the friction-factor of the material they are moving on.

A collision detection prevents the robot from going through walls or other objects. Another good feature would be simulation of the object dynamics.

At the Eindhoven University of Technology (Netherlands)[1] are two packages available which could be used to do this job.

[1] `www.tue.nl`

The first one is SOLID[2] by Gino van den Bergen which realizes a collision detection, and the other is DYNAMO[3] by Bart Barenburg which can handle dynamics of objects. There you can also find some links to related pages.

The Architecture of the simulator has to be thought over. In it's current state it seems neither to be very consistent, nor even easy to extend.

[2] www.win.tue.nl/cs/tt/gino/solid/
[3] www.win.tue.nl/~bartb/dynamo/

Chapter 8

Conclusion

The project was originally planned to produce contols for examining the Khepera robot. Therefore we got a simulator (EasyBot) and the real robot. The first step to do should be to try a control using EasyBot before porting it to the robot. After a few tests we found out, that the porting from EasyBot to the real Khepera robot meant to design a completely different program. This was caused by major differences between EasyBot and the real robot.

Therefore our main objective changed to build up common interfaces to have one control for all kinds of using. We intended to generate a robotics programming interface, which allows to develop a control that is testable independendly from the real robot. After testing on the simulator, it should work similary on the real robot.

The project was successfully in a way that we designed a structure to program a Khepera robot in a fairly easy way. The control just need to be written once to be executable on the simulator, online (via the serial connection) and directly on the robot. Due to the lack of time we were not able to implement the whole functionality for the simulator. Caused by using C++, there are still problems with the cross compiler, which does not work properly.

The secondary aim was to generate a set of tools, which are necessary to work with the robot (serial communication, download . . .). When we wrote these tools we decided to have tools for Linux and WindowsNT.

We produced a few controls to demonstrate, how to use the simulator. Some of them does not fit to our interface. These controls were necessary to find advantages and disadvantages of the simulator and our interface.

After providing this interface and these tools, now its time to write controls to use them.

Have a lot of fun !

Appendix A

Directory Structure

If there is a `zip`-file in a directory, the extracted content of this file is also provided in a subdirectory of the same name. In most cases you should use the `zip`-file.

Khepera /

- **Documentation /** contains this documentation
- **EasyBot /**
  - **controls /** example controls for EasyBot
  - **framework /** framework for a simple DLL for EasyBot
  - **interfaces /**
    - **Khepera /** common Khepera interface as described in chapter 3
      - **bioscom /** implementation for cross compilation (does not work yet) – see section 5.1
      - **examples /**
      - **img /** images needed for running qserial_...
      - **include /** header-files for the common interface
      - **netcom /** serial port implementation
        - **commport.unx /** Linux communication on serial port
        - **commport.nt /** WindowsNT4 communication on serial port (does not work yet).
        - **qtdir /** Qt-frontend for serial communication on Linux
      - **simcom /** implementation for the simulator
        - **commport.nt /** dummy serial port for the interface
        - **DLL_SimCom /** Programming interface for a control-DLL see section 3.3.2
        - **LIB_SimCom /** sources to generate a new library used in `DLL_SimCom`
  - **src /** sources for writing a DLL as described in section 2.3

Khepera / *continued*

- **EasyBot /**
  - **extensions /** extensions for the simulator
    - **RayInfo /** the RayInfo-Control as described in section 2.4
  - **include /** header-files for EasyBot-controls
  - **release /** precompiled controls for EasyBot
  - **worlds /** LV3D-scenes for robot simulation
- **External /** programs and tools from external sources
  - **Doc /** Documentation of the Khepera Team (Lausanne)
  - **Examples /**
  - **Papers /** scientific papers about the Khepera robot and algorithms for it
  - **Tools /**
  - **Webots /** Webots robot simulator
- **LV3D /** the 3D-viewer LightVision3D
- **Tools /**
  - **upload /**
    - **Linux /** the Khepera Upload tool for Linux as described in section 6.1.1
    - **WinNT4 /** the Khepera Upload tool for Windows NT 4 as described in section 6.1.2
- **Presentations /**
  - **Paderborn /** our material for the “First international Khepera Workshop” in Paderborn, 10th / 11th December 1999.

Bibliography

[1] Hochschule für Technik und Wirtschaft Dresden,
Friedrich-List-Platz 1, 01069 Dresden
`http://www.htw-dresden.de`

[2] Prof. Dr. Heino Iwe
`http://www.htw-dresden.de/~iwe/easybot/easybot.html`

[3] LightVision3D
`www.lightgraphx.de`

[4] Khepera User Manual – *Version 5.02*
K-Team, Ch. de Vuasset, CP 111, 1028 Préverenges, Switzerland
`http://www.k-team.com`

[5] Khepera - GNU C Khepera cross-compiler installation instructions K-Team, Ch. de Vuasset, CP 111, 1028 Préverenges, Switzerland
`http://www.k-team.com`

[6] K213 Vision Turret User Manual – *Version 1.2*
K-Team, Ch. de Vuasset, CP 111, 1028 Préverenges, Switzerland
`http://www.k-team.com`

[7] Stuttgart Neural Network Simulator
`http://www-ra.informatik.uni-tuebingen.de/SNNS/`