robbit v3.0 documentation

a graphical simulator for multi-robot formations

Chitresh Bhushan and Sayandeep Purkayasth

December 7, 2007

This text is a brief description of the features that are present in the robbit version 3.0, 08 December 2007. This is Edition 2.0, last updated 08 December 2007, of robbit documentation, for robbit, version 3.0.

The **people** involved in this project are listed below with contact details.

Antonelli Gianluca
Professor
Dipartimento di Automazione, Elettromagnetismo
Ingegneria dell'Informazione e Matematica Industriale
Università degli Studi di Cassino, Italy
antonelli@unicas.it
http://webuser.unicas.it/antonelli

Arrichiello Filippo
Research Assistant
Dipartimento di Automazione, Elettromagnetismo
Ingegneria dell'Informazione e Matematica Industriale
Università degli Studi di Cassino, Italy
f.arrichiello@unicas.it
http://webuser.unicas.it/arrichiello

Bhushan Chitresh
Undergraduate student
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology Kharagpur
chitresh.bhushan@gmail.com
http://chitresh.co.nr

Prakash Chander
Undergraduate student
Department of Computer Science Engineering
Indian Institute of Technology Guwahati

Purkayasth Sayandeep
Undergraduate student
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology Kharagpur
deepcyan@gmail.com
http://deepcyan.co.nr

The **project website** is http://webuser.unicas.it/robbit.
It is also available on Sourceforge: http://sourceforge.net/projects/robbit/.

Contents

1 Introduction

1.1 Concerning robbit

With so much research going on concerning robots in this world today, and so much more on wireless networks, we decided that what this world really needs is a good platform for testing out motion algorithms for robot formations. Now we all know how diffcult it is to get a bot moving how you want it to, so we thought a simulator ought to do the trick. A 3D simulator: all the better. And so it was. With OpenGL in one hand and OpenCV in the other, we set out to do what we were born to do: robbit.

I guess a wee bit on the nomenclature would do some good at this point. Well it so happens that just before starting out on this wonderful venture, one of us was highly impressed by the LOTR series especially by resilience of some good-natured hairy-footed creatures called $hobbits$. And viola! $robbit$ it was.

1.2 A bit more

Professor A. Gianluca was looking into various motion algorithms and testing them out on real robot MANETs (Mobile Ad-Hoc NETworks). These were contructed so that an algorithm decided each robot's coordinates according to some predefined objective. For example, we have 5 robots that are to surround a randomly moving ball, then follow it. Now various algorithms are possible and testing each of them on the physical plane is quite hectic, believe me. So he decided that he needed something on a simulated environment. Thats where we came in.

So we started up with some sample codes and created a 3D environment consisting an experimentation plane some moveable robots, a ball, and some lighting. Gradually we progressed from keyboard view control to mouse control, from a dumb pink floor (what were we thinking then?!) and box-like robots to a shiny chessboard floor, from crappy to perfect snapshots and even video recording. Now you could even place obstacles on the chessboard to see if your robots can see them.

2 How to use this manual

2.1 Conventions

We'll be following some conventions in the part of the manual that follows. They are noted in Table 1.

Text style	What you're looking at
`text`	a file name
text	a function name
text	a variable name
TEXT	a linux package name

Table 1: Formatting conventions used in manual

3 Installation

3.1 Prerequisites

We used the following libraries as part of our project.

- OpenGL
  The Intel **Open G**raphics **L**ibrary is a set of data structures, and functions to implement 3 dimensional scene rendering.
- GLU
  The Open**GL U**tility Library has some routines that provide higher-level drawing routines from the more primitive routines that OpenGL provides.
- GLUT
  The Open**GL U**tility **T**oolkit is a library of utilities which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, monitoring of keyboard and mouse input, and drawing some geometric primitives.

  Together OpenGL, GLU, and GLUT are used in game graphics, etc. Here they serve as the foundation for rendering 3 dimensional objects like the robots, obstacles, the floor, the ball, etc. They also provide facilities for GUI, windowing, etc. which we used to our advantage.
- OpenCV
  The Intel **Open C**omputer **V**ision Library is a set of data structures, and functions for image handling and processing. We used its features of image saving and video recording to provide researchers a output format more appealing than boring text files containing infinite coordinates, velocities, orientations, etc.

The above prerequisites may be obtained and installed as follows.

- Windows systems
  - C/C++ compiler
    We used Microsoft Visual C/C++ 6 compiler. We have provided the workspace we created in the package under `source/win32/Robbit.dsw`. This may be used directly for further development.
  - OpenGL, GLU, GLUT
    We have provided the GLUT installation package along with our distribution. It maybe found under `libraries/glut/`. Installation instructions are present in `libraries/glut/README`
  - OpenCV
    This may be obtained from SourceForge®. A windows installer is available, which provides the basic header files, dynamically linked libraries, etc.

    Please read `openCV instructions.pdf` in case you plan to set up your own workspace, in which case, you will need to configure it to find required files from the OpenCV installation.
- Linux/UNIX systems
  - C/C++ compiler
    We used GNU C compiler GCC4.1.2 (GCC), which is available from http://www.gnu.org.
  - OpenCV
    The source code from this library is available at SourceForge®. Note that since video support is essential for our project to be fully functional, it is necessary that you build OpenCV from sources with FFMPEG (a library of audio/video codecs) support. Please read OpenCV installation instructions (available along with the source package) especially the part dealing with libavcodec (a part of FFMPEG) support. For the same reason, it is recommended that OpenCV binaries (LIBCV0.9.7-0, LIBHIGHGUI0.9.7-0, LIBHIGHGUI-DEV, LIBCV-DEV or their later releases) not be used, unless the developer is sure that FFMPEG support is built into the binary.

– OpenGL, GLU, GLUT
The linux packages GLUTG3, GLUTG3-DEV must be installed along with their dependencies, including hardware dependent drivers. A package manager like SYNAPTIC will be able to tell you these packages and install them for you. To test OpenGL installation, one may use a program like GLXGEARS.

3.2 Installation steps

The source may be obtained from the project webpage: http://webuser.unicas.it/robbit or SourceForge: http://sourceforge.net/projects/robbit/.

- Windows systems
After setting up the prerequisites, and adding OpenCV folders to the project include directories, simply open the workspace, compile `Robbit.cpp`, and build a Release of the project. The final executable shall then be available in `source/win32/Release`.

- Linux/UNIX systems
A makefile, `Makefile` has been provided that takes care of the compilation, linking, etc. Use the command below.

```
$ make Robbit
```

The binaries are produced and moved into the directory `bin`. Add this directory to the PATH variable if you may require frequent use of the binary.

4 Features

We have endeavoured to make the interface as user-friendly and intuitive as possible. Apart from expected features like pause/play, replay, exit, increase/decrease speed, we have provided the following features.

1. Changing camera location, viewing direction and zoom
   (a) intuitive mouse controlled movement of the camera
   This is easier to control than a keyboard control which involves delay and low resolution of movement.
   (b) options for viewing from positions directly above the board (Top view), and from just above each robot
   These allow close observation of the arena and may help improvement of the algorithm under test.
   (c) Zoom in/out

2. Save snapshot
   This function reads pixels from the frame buffer and processes them into a JPEG image using built-in OpenCV library functions. Images are saved in the working directory with the following file name format.

   `capture_<time_int>_<theta>_<phi>.jpg`

   Here `theta`, `phi` refer to θ and ϕ of the OpenGL camera position with respect to the spherical coordinate system.[1] `time_int` refers to the time (in integer casted form) when the snapshot is taken.

3. Save video
   This function uses built-in OpenCV library functions to write frames to an AVI file. The codec to be used can be selected in Windows (A pop-up window shows the available codecs for video compression. One may be selected.) whereas it is restricted to `DIVX` codec in linux environment. Default *frames per second* (fps) is 25 fps. Videos are saved in the working directory with the following file name format.

   `capture_<start_time_int>_<theta>_<phi>.avi`

   Here `start_time_int` refers to the time (in integer casted form) from which video capture starts. `theta`, `phi` again refer to θ and ϕ are the respective values of the start frame view. Please note that enabling video recording reduces rendering speed and performance in general, due some inherent delay in the used OpenGL function to capture the screen. This feature is currently under some investigation due to some bug reports.

4. Info Box
   This is a sub window (within the animation window) that shows various runtime information. It shows the last command passed, the zoom level, θ, ϕ, etc.
   Please note that enabling enabling Info box option also reduces rendering speed and performance, in general due some inherent delay in rapidly rendering changing text.

5. Obstacles
   We have included the facility of allowing the user to define his own testing environment by placing obstacles of arbitrary dimensions as required. For this purpose, all such obstacle information are to saved in a file `Obstacle.txt` in a specified format noted in the file itselfin its commented part (lines starting with '%'). A sample obstacled environment is given so that the user may understand the format easier.

6. Coordinate generation
   Earlier, we relied solely on an input file, the filename of which was stored in $input\_file\_name$ in `Definitions.h` which is by default, set to `output_pos.log`. This file contains the coordinates of the various robots and ball at many time samples in a specified format mentioned in its commented part (lines starting with '%').
   However now the user also has the option of adding his own algorithm into the source code itself, so he may test it without first generating an intermediate file. This is achieved through 2 sub-choices.

[1] $x = \rho \sin\phi \cos\theta$, $y = \rho \sin\phi \cos\theta$, $z = \rho \cos\phi$. $\rho = 250 units$ has been set constant at the start of the animation.

- You may opt for an online implementation of your algorithm. Here you must put your algorithm inside function *GetNextFrame()*. You are expected to set all the values of the elements of the frame object *current*. An example is provided for reference. Note that CPU intensive algorithms may slow down the simulation, making it appear paused for short durations. In this respect, the next option is more preferable.
- You may opt to create an input file of the form of `output_pos.log` (supplied) using your own algorithm. However all writing operations must be stopped before the simulation starts. Here you are expected to put your algorithm to generate the input file strictly in the format shown in `output_pos.log` inside the function *WriteInputFile()*.

7. Trails
   We have provided for tracing of each objects motion path using the concept of trails. This feature may be switched off if required.

8. Median of robots
   We have also rendered a point on the Experimental plane equivalent to the median of the positions of all the bots. In the experiment suggested in the introduction, the distance between this median and the ball may serve as a measure of the efficiency of the algorithm.

9. Support for different robot designs
   We have added a choice between two hardcoded designs **KheperaII** and **KheperaIII**. A developer may added more designs if required.

10. Collision detection
    We have added a support for detection of collision of the robots with the ball, the obstacles and other robots. The robots change color to red, green and yellow on colliding with another robot, the ball and an obstacle, respectively.

5 Usage

5.1 Code flow

The present code executes in the following sequence.

1. On running the executable, the user is first asked which of the following modes of frame generation should be followed.
   (a) Use existing `output_pos.log` file for coordinate data input.
   (b) Generate cooridnates for next frame using a function GetNextFrame(). This contains the algorithm for generating coordinates, velocities and orientations of the robots at any instant with/without previous frames' information. The function presently contains a sample code which may be replaced by a user with a more practical code.
   (c) Generate new `output_pos.log` using a user-defined algorithm stored in WriteNewFrame() and then use it for simulation
2. Obstacles if any are read into memory from the file `Obstacle.txt` by function ReadObstacle().
3. Then the user is asked which design (Khepera II or Khepera III) is to be used for simulation.
4. The simulation is started. Now the user may control the simulation using keyboard shortcuts noted in Table 2 or using the right-click menu.

Shortcut	Action
Animation control	
<escape>	Stop and Exit the simulation at any time
<space>	Toggle pause/play of simulation
r	Replay animation from start
View control	
0	Return view to isometric view
t	Change view to Top view
s	Take a snapshot of the simulation
v	Toggle video recording of the simulation
+/-	Zoom in/out
<number>	Shift OpenGL camera on to top of robot numbered `number`
Accessories	
i	Toggle display of Information box
o	show/hide obstacles
n	Toggle numbering of the displayed robots

Table 2: Runtime keyboard shortcuts. Note: Uppercase forms of the specified characters may also be used with the same effects.

5.2 Right-click menu

All the keyboard options have been included in the right-click menu. Over and above these, the following animation options are also available.

1. Light control
   Options for increasing number of enabled lights up to a maximum of 4 lights have been provided. The light source positions, color and other parameters are configurable in `Definitions.h`

2. Animation speed control
   Options for increasing speed of animation from 1X up to a maximum of 25X have been provided.

3. Robot design
   At the moment, two designs are hardcoded and selectable. These are that of KheperaII and KheperaIII. Selection of design may be done before start of simulation or at runtime (using right-click menu).

4. Trail display
   The user may opt to switch off display of trails during runtime.

6 Development

6.1 Data structures

We created a class *frame*, containing the following elements and methods.

Data Type	Variable/Function name	What it does
float	*time*	current time of the frame
float	*time_step*	current delay time (for sleep functions)
int	*bot_design*	1: Khepera II; 2: Khepera III
float	*bot_x [no_of_bots]*	x-coordinate of the robots' position [cm]
float	*bot_y [no_of_bots]*	y-coordinate of the robots' position [cm]
float	*bot_vx [no_of_bots]*	x-component of the robots' velocity [cm/sec]
float	*bot_vy [no_of_bots]*	y-component of the robots' velocity [cm/sec]
float	*bot_orient [no_of_bots]*	orientation of the robots [rad]
float	*bot_vorient [no_of_bots]*	angular velocity [rad/sec]
int	*bot_hit [no_of_bots][2]*	*bot_hit[<bot_number>][0]*: not hit (0) or hit (1) *bot_hit[<bot_number>][1]*: time of hit for each bot
float	*bot_center_x*	x-coordinate of current cetroid of the robots
float	*bot_center_y*	y-coordinate of current cetroid of the robots
float	*ball_x*	x-coordinate of the balls' position [cm]
float	*ball_y*	y-coordinate of the balls' position [cm]
float	*ball_vx*	x-component of the balls' velocity [cm/sec]
float	*ball_vy*	y-component of the balls' velocity [cm/sec]
(methods)	*frame()*	constructor
	update(int mode, ifstream fp)	reads input file and updates object data
	render_frame()	renders the frame data on using OpenGL functions

Table 3: Elements of class *frame*

It contains all the information required to render a frame. This information (as discussed earlier, may be obtained from the input file `output_pos.log` or using an algorithm (directly or indirectly).

6.2 Code organisation

The source code is organised as follows. Each function and method, `function_name()` is contained in its separate header file named `function_name.h`. The filenames and an overview of the work of the corresponding functions are noted in Table 4. The file `Robbit.cpp` contains the main code to be compiled that uses these header files.

File	What it does
BotHit.h	Determines which robot has been hit and marks it
Definitions.h	Animation parameter definitions & includes
DetectObstacleCollision.h	Detects the collision beteween any robot and any obstacle
Display.h	Main function which draws all the things in main window
DistancePointLine.h	Returns if any robot has collided with a wall
DrawFloor.h	Draws the floor on which robots are moving
DrawObstacle.h	Renders the obstacles given in the ASCII file
DrawRightClickMenu.h	Draws the right click menu
DrawString.h	Draws 2D-text in the window
DrawTrails.h	Draws trails of robots and ball
FrameRenderFrame.h	This renders all elements of the frame object in the simulation window
FrameUpdate.h	Reads from the ASCII log file and creates the current frame
GetNextFrame.h	Creates the object for the next frame based on a user-defined algorithm
Init.h	Initilization of the main window
KeyEventHandler.h	Monitors the keyboard input
KillAnimation.h	Kills the animation at any time as required
MenuSelect.h	Monitors the right click menu Select
Motion.h	Rotate the scene in 3D with the left mouse button
Mouse.h	Monitors the mouse clicks
NextNo.h	Takes a string and returns the next identifiable number
OutputCharacter.h	Draws 3D-strings in the desired location (x, y, z) in window
PlayControl.h	controls the animation (speed, begin & end)
PositionKhepera2.h	Draws the KheperaII Robots with all the specifications
PositionKhepera3.h	Draws the KheperaIII Robots with all the specifications
ReadObstacle.h	Sets the corresponding Obstacle parameters from
Reshape.h	Determines the Eye/camera location
RobotClass.h	Contains the definitions of class `frame`
SubDisplay.h	Draws the sub-window (info-box)
SubReshape.h	Reshape for the Sub-Window
VideoDump.h	Saves the current running simulation as video in AVI format
WindowDump.h	Saves an animation snapshot (Screenshot)
WriteInputFile.h	Write the input file `output_pos.log` using user-defined algorithm
Robbit.cpp	Intializes the main simulation window & its parameters

Table 4: The files involved and what they do