Graphical Improvements to RobotBuilder

Presented in Partial Fulfillment of the Requirements for
The Degree of Master of Science in the
Department of Electrical Engineering

By

Siswanto Hartono

* * * * *

2003

M. S. Committee:

Dr. David E. Orin, Adviser
Dr. Charles E. Klein

ABSTRACT

RobotBuilder is a graphical software application that can be used to rapidly develop robotic dynamic simulations. It has the familiar and easy to use Windows graphical user interface. The program also includes RobotModeler, a simple application to develop graphical models for the RobotBuilder. The flexible control design in RobotBuilder has made it useful to be used in a range of tasks from introductory robotics education to advanced robotic control research. This program package is developed at Ohio State University by Steven Rodenbaugh. Luke Frankart then implemented additional features to Steven's RobotBuilder. His added features primarily relate to the camera control. He changed the code so that user view is maintained between Examine and Pan Mode camera control. He also added a vertical stabilizer to prevent the terrain from moving up and down during simulation. A tutorial to facilitate the creation of robots using the RobotModeler was also written using the Whegs robot as a model. This project builds upon those works by adding more functionality to RobotBuilder. Specifically, additional features for the camera control and the terrain. The user can now specify the view angle of the camera. A separate X and Y grid resolution for the terrain is created for a more flexible terrain sizes. Also, a wireframe overlay on the terrain feature is added that may prove useful for some users.

ACKNOWLEDGMENTS

I wish to thank Dr. David E. Orin for all of his guidance and support throughout this research. I am also grateful to Dr. Charles E. Klein for his participation on the examination committee. I would also like to express my gratitude to Steven Rodenbaugh for his support and insight into RobotBuilder's algorithm. I would like to thank Luther Palmer for his thoughts and advice, and allowing me to use his computer to do this research. My gratitude also to Wei Hu for adding the support for a separate X and Y grid resolution in the loading terrain data function in *DynaMechs*.

Finally, I would like to thank my entire family for their continuous support throughout this research and my entire education.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

RobotBuilder [1] is a graphical robotic simulation application with an easy to use Windows graphical user interface (GUI). It is basically a graphical envelope for the *DynaMechs*, a C++ library developed by Scott McMillan [2],[3]. The GUI was implemented using the *Microsoft Foundation Classes* (*MFC*). *WorldToolKit* (WTK) graphics library was used to implement the three dimensional graphics in the scene [4]. Additional features to RobotBuilder were then implemented by Luke Frankart [5]. He added a more flexible camera control so that user view is maintained between Examine and Pan Mode camera control. He also added a vertical stabilizer to prevent the terrain from moving up and down during simulation. A tutorial to facilitate the creation of robots for RobotBuilder using the RobotModeler was also written using the Whegs robot as a model.

However, the robots in the simulation do not appear really realistic. An example of a nicer looking graphics is shown in *GLAnimate* demonstration of WAAV (wheeled, actively-articulated vehicle) by Min-Hsiung Hung [6]. Consequently, a more realistic looking robots in the RobotBuilder scene are desired.

1.2 Objectives

This project aims to improve the graphical appearance of the robots and the scene in RobotBuilder in part by adding additional functions such as the camera's view angle control, a separate X and Y coordinate grid resolution of the terrain, a wireframe overlay on the terrain, and a better lighting model. In addition, a start-up picture is to be added to improve recognition to RobotBuilder.

1.3 Organization

Chapter 2 covers the implementation of the new features implemented in RobotBuilder. Some motivations for these features are discussed along with the implementation details.

Chapter 3 contains a discussion of the algorithm implemented, and is intended to help future programmer to the RobotBuilder understands the features implemented in this version release.

Chapter 4 contains a summary of the research completed and suggestions for future enhancements.

The appendices contain the new file formats for the configuration (.cfg) files and terrain (.dat) files.

CHAPTER 2

ENHANCEMENTS TO ROBOTBUILDER

2.1 Background

The objective of this project was to implement new features in the RobotBuilder program to give the user more control over the scene. The first area for the additional feature is in camera view. A view angle degrees selection is to be added so that users may select if they want to be able to see more of the perspective effects on the scene or not. A more exaggerated perspective effects on the scene can make it appear more realistic. Two new features are also added in the terrain data parameters to improve RobotBuilder's performance when using slower computers, and enhance the terrain's appearance. A new lighting model is to be added to have a better graphical animation of the robots, especially the Whegs demo. Currently, each time RobotBuilder is executed, the main window only displays the inertial axes and background color. With a startup picture added, the user will be welcomed by a RobotBuilder logo instead of a plain colored screen.

2.2 Adjusting the Camera's View Angle

RobotBuilder were originally programmed with a fixed view angle of 9°. With such a narrow view angle, the perspective effect on the scene is not visible. In essence,

the perspective effects in the scene can be changed by setting the view angle. With a wider view angle, the perspective effects can be more noticeable in order to simulate a real-life perspective. However, with a wider view angle, the object will appear farther away and smaller in the scene. Figure 2.1 below illustrates this relationship.

By default, the scene is set to a narrow view (9°) so that previous configurations do not change in appearance when opened using the new version of RobotBuilder. The user can then set a wider view angle, for example 36°, so that the perspective effect is much more visible as opposed to before. The user can also save this view angle setting in the new version of the configuration (.cfg) file, which will be recalled when the newly saved configuration file is opened.

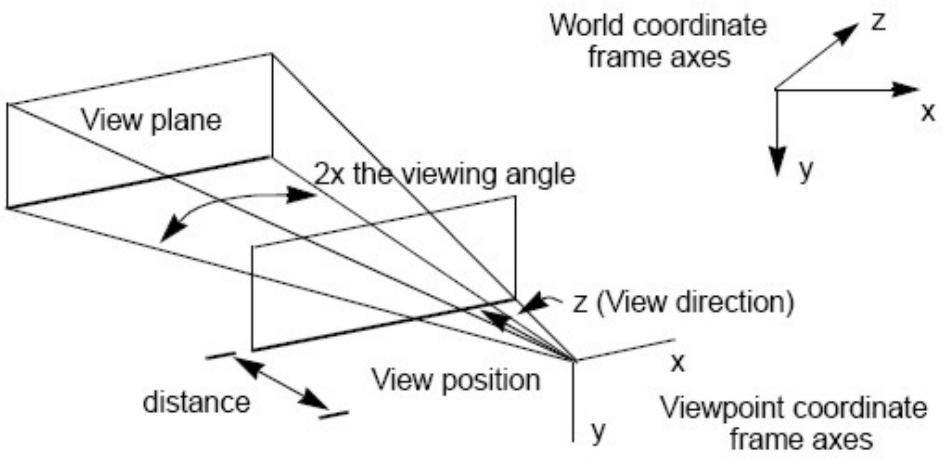

Figure 2.1: View angle Illustration

This feature is put inside the Camera Control dialog under the “View” menu. Figure 2.2 shows the modified dialog box. The user can type the view angle they want into the edit box and the scene will be updated correspondingly. Setting an extremely

narrow or wide angle produces a very distorted scene and is not very useful, that is why the recommended view angle range is 5° –75°. Take note however that with a wider view angle, the object will appear farther away and smaller in the scene. Conversely, the object will be very close if changing to a much narrower view angle.

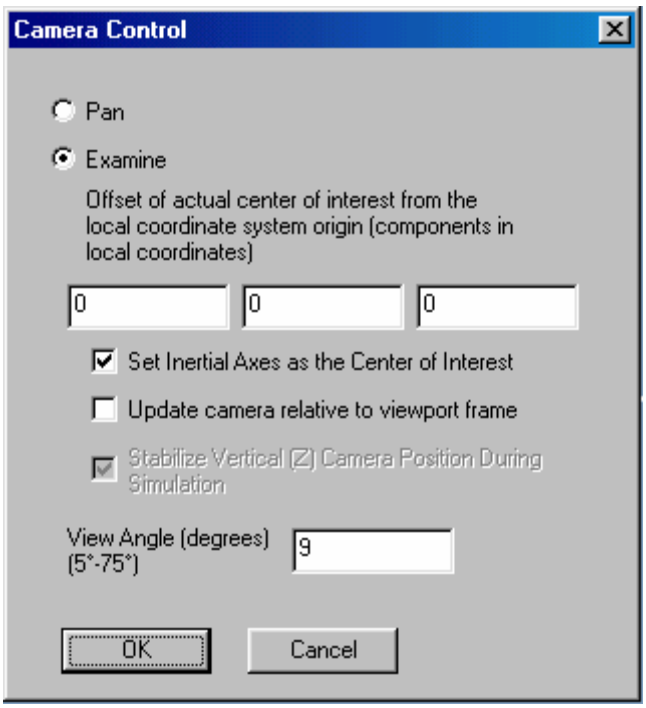

Figure 2.2: Modified Camera Control Dialog Box

2.3 Separate X and Y Grid Resolution for the Terrain

Previously, the terrain only has one grid resolution for both the X and Y coordinate. A large data points for the terrain may causes the program to be unresponsive when run on lower-end computers. An example to this is the Whegs demo terrain. To alleviate the burden on the lower-end machines, the user can reduce the dimension of just one coordinate, and still have large enough terrain by increasing the resolution of that

coordinate. Figure 2.3 below shows the new Terrain Data dialog box. This setting can also be saved in the new version of the terrain file.

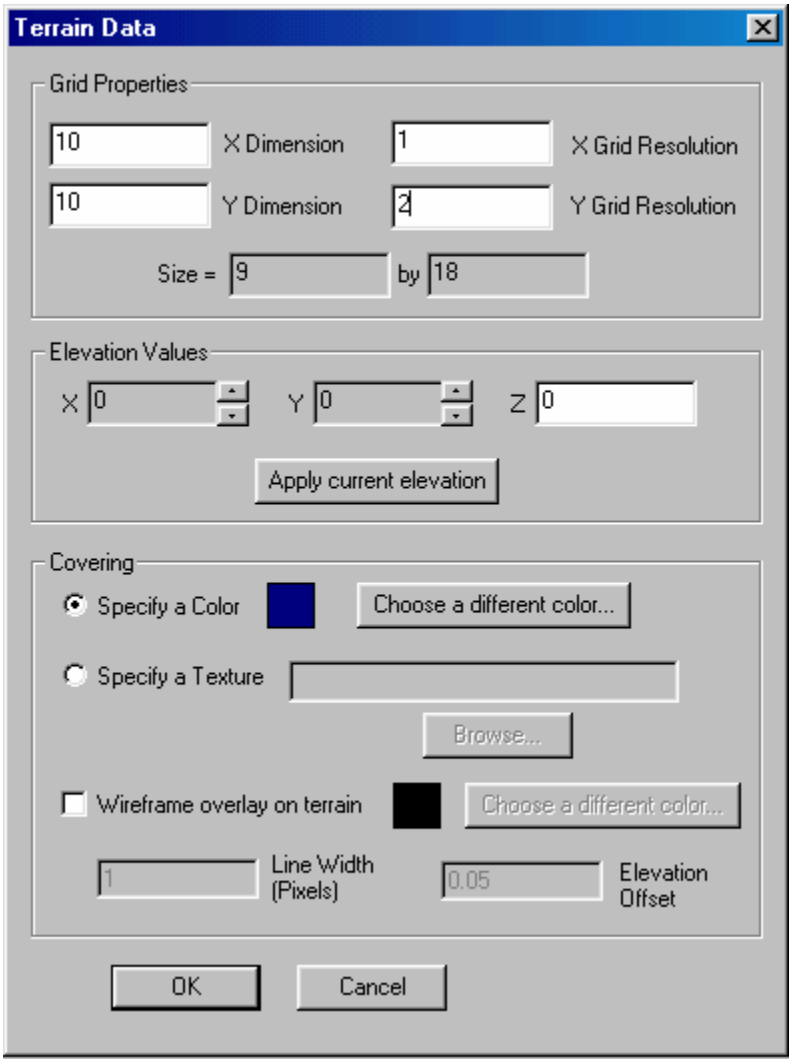

Figure 2.3: Modified Terrain Data Dialog Box

2.4 Triangle Wireframe Overlay on the Terrain

A supplementary feature that some users may find useful is to have a triangle wireframe on the terrain. A checkbox for this is added in the Terrain Data dialog box. This wireframe overlay on the terrain can help to differentiate the terrain polygons. An example to the Cart project is shown in Figure 2.4 and Figure 2.5.

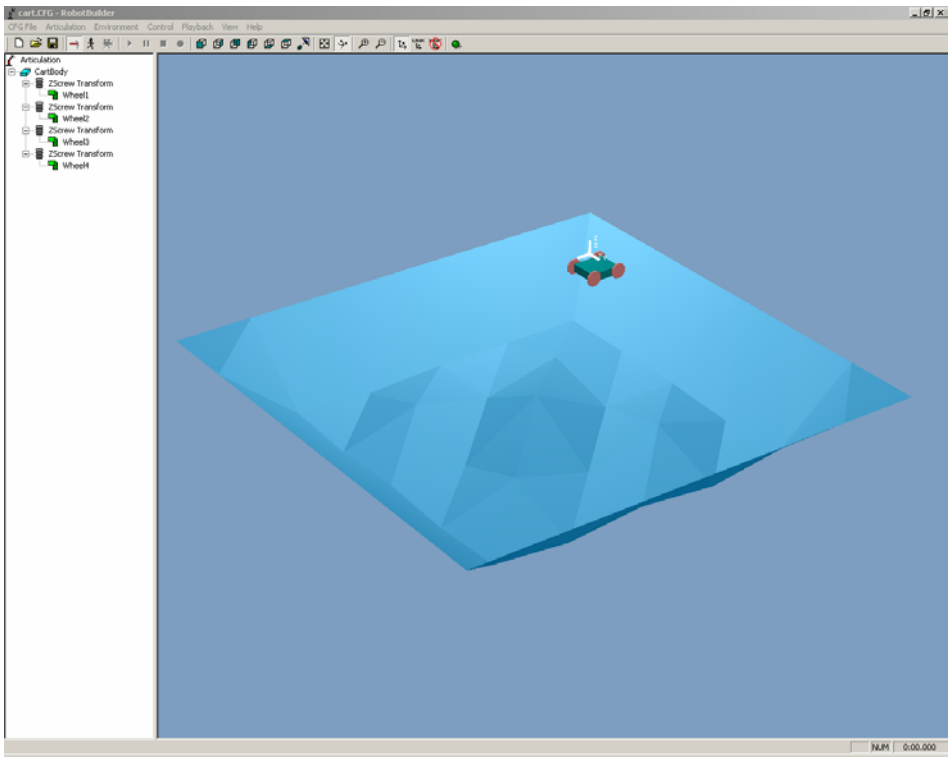

Figure 2.4: Cart without Wireframe Overlay

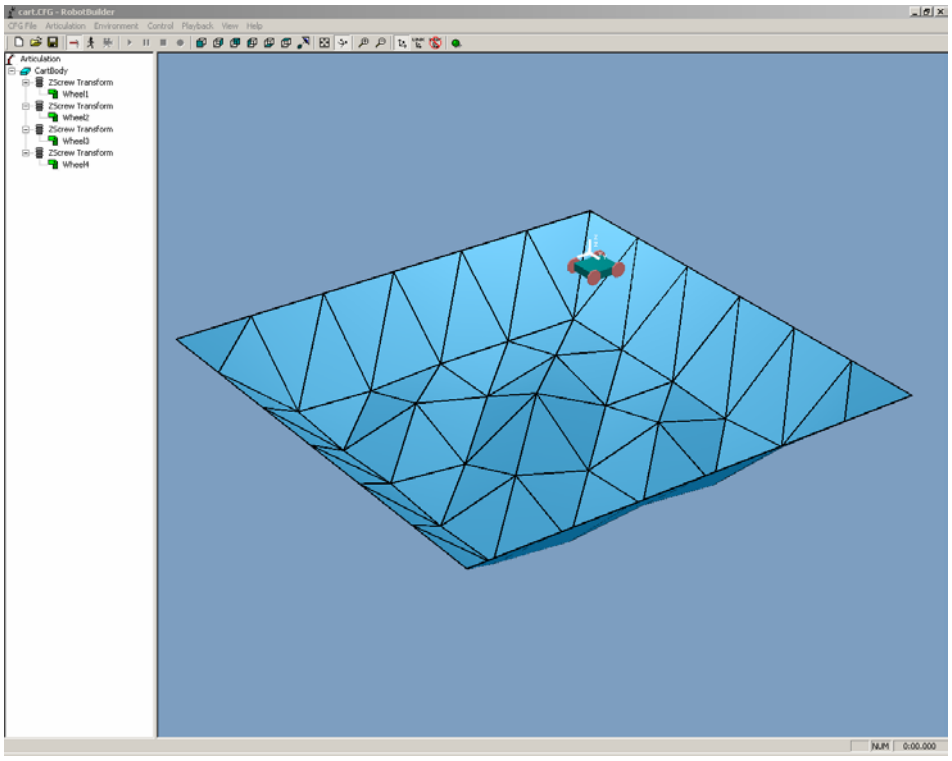

Figure 2.5: Cart with Wireframe Overlay

As shown in Figure 2.3 before, the user can specify some parameters of the wireframe overlay. The default parameters for the wireframe are black, 1 pixel width, and 0.05 elevation offset above the terrain. The color can be changed by clicking “Choose a different color...” next to the Wireframe overlay checkbox. It will then open a color selection dialog box where user can choose the color or specify the RGB values of the color. Figure 2.6 here shows a sample color dialog box.

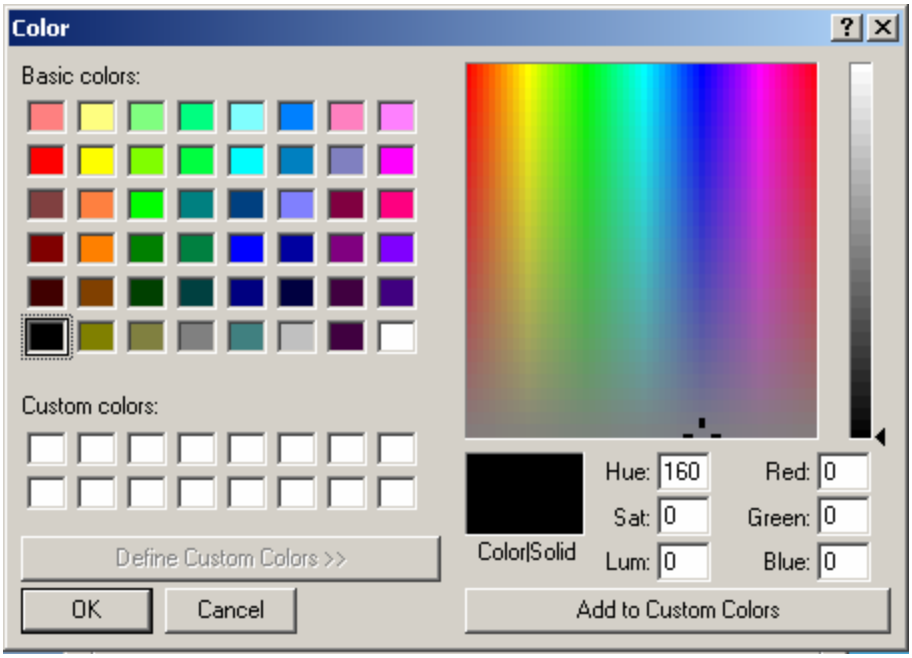

Figure 2.6: Sample Color Dialog Box

The two other parameters can be specified by typing the desired values in the respective edit boxes. The reason to have an elevation offset is because of Z-buffering issue. This is when the graphic card cannot decide which item is on top of which and thus may cause some flickering of the lines. The offset prevents these missing lines. The drawback in turning on this wireframe overlay is that it may dominate the terrain when viewed from afar and the terrain has small polygons.

2.5 Better Lighting Model

A slightly better lighting is achieved by adding a point light in the RobotBuilderLights.wtk light file. By adding this light, the highlights and shadows of the robot surfaces are more prominent and thus looking more realistic and not flat. Figure 2.7 and Figure 2.8 contrast the results on the Whegs robot.

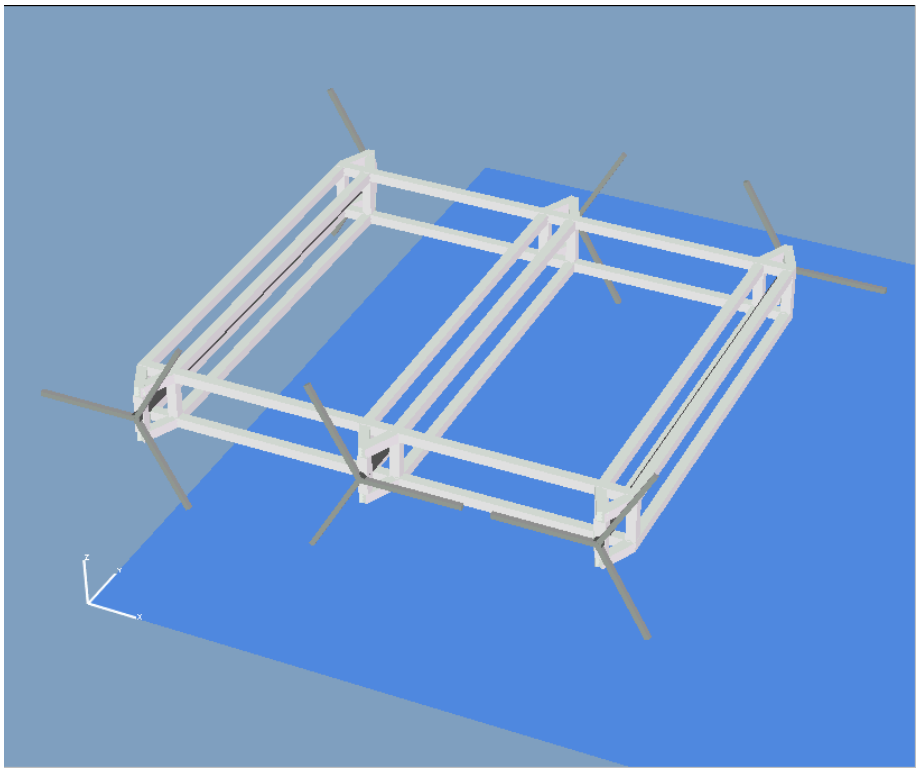

Figure 2.7: Whegs without a Point Light

The Whegs robot with an added point light in the scene has a more pronounced highlight and shadows on its body. It is easier to see the square frames of the Whegs body with an added point light in Figure 2.8.

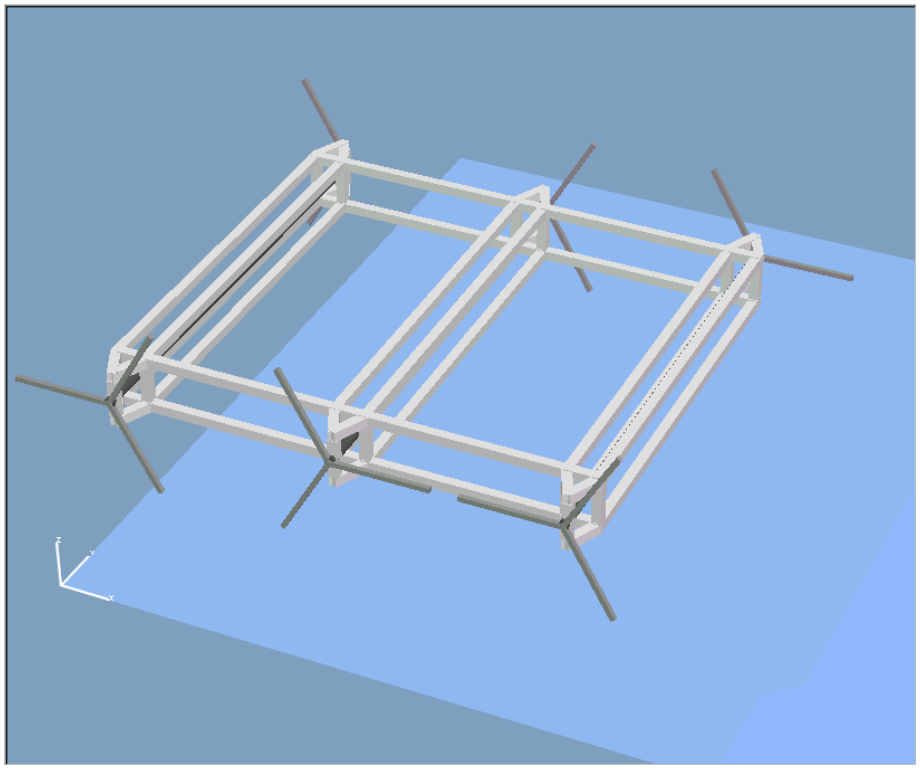

Figure 2.8: Whegs with a Point Light

2.6 Startup Logo Picture

A startup picture that is shown when RobotBuilder is run initially would be beneficial in promoting RobotBuilder. This picture can serve as some kind of a logo to the software package. It may even increase the software's recognition. Increased recognition of the program could really bring in more funds or grants to the department. Figure 2.9 shows a sample of the startup picture in RobotBuilder.

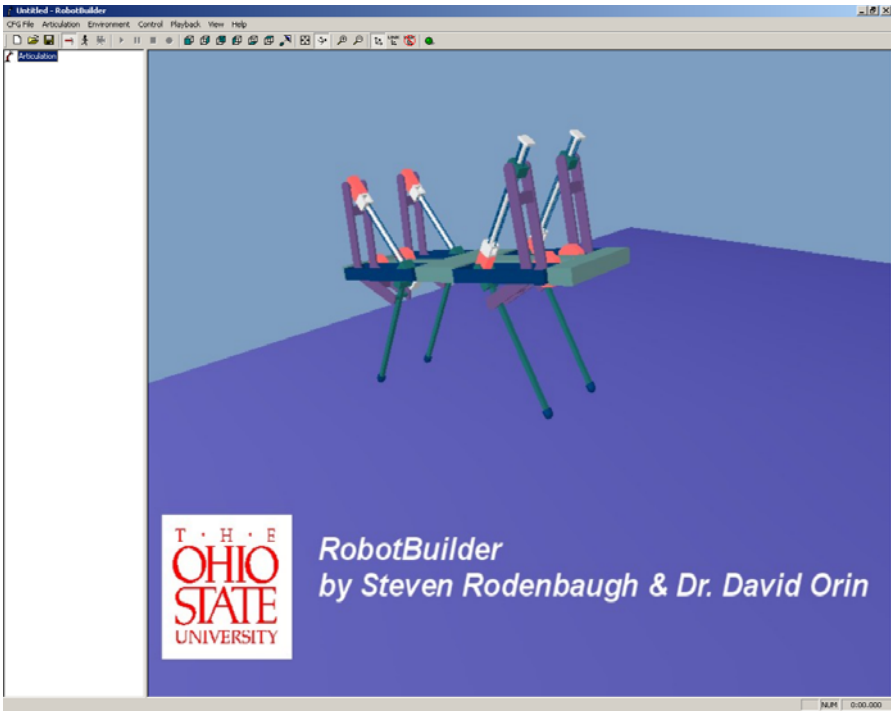

Figure 2.9: Startup Image

However, unsuccessful debugging of this implementation prevents it to be included in this version release of RobotBuilder, version 1.1.

CHAPTER 3

PROGRAMMER'S GUIDE

3.1 View Angle

The perspective effect in the scene can be changed by setting the view angle, which is done in the WTK window by using the function *WTwindow_setviewangle*. This function sets the WTK's window horizontal view angle. The vertical view angle is automatically sets as well, based on the dimensions of the window. The horizontal view angle is defined as half the horizontal angular field of view (in radians). This function has no effect if the angle supplied to this function is not between 0.0 and PI/2.0 (90°), exclusive. The GUI for Camera Control under the "View" menu is also updated to include the edit box for the view angle where the user can input the angle. See Figure 2.2 for the new dialog box. The following files were revised for this implementation: DlgCameraControl.h/.cpp, CameraData.h/.cpp, dmvCFGData.h/.cpp, ParserCFG.h/.cpp, GenerateCFGFile.cpp, and DMViewerDoc.cpp.

In order to be able to save the view angle value, a new version (Version 3) of the configuration (.cfg) file was needed. The new configuration file format is shown in Appendix A. The new parser was written so that the program knows which version of the configuration file is specified and thus ensuring backward compatibility with older configuration files. It follows that the configuration file generator is also updated. In addition to this, there are additional internal data structures in CameraData and dmvCFGData to accommodate the view angle variable storage.

3.2 X and Y Grid Resolution for the Terrain

To alleviate the problem of RobotBuilder's unresponsiveness on lower-end computers, a separate X and Y resolution for the terrain is implemented. In order to do this, the *DynaMechs* package used for the dynamic calculations also needs to be revised. Specifically the function *loadTerrainData* in the dmEnvironment file. Previously, that function only takes 4 parameters: x_dimension, y_dimension, grid_res, and depthArray. The new overloaded function written by Wei Hu is able to take 5 parameters: x_dimension, y_dimension, x_grid_res, y_grid_res, and depthArray. To be able to use this new function, *DynaMechs* has to be recompiled so that the necessary libraries used by RobotBuilder can be updated. Otherwise, there will be linker errors when compiling RobotBuilder. The implementation of this function is used by RobotBuilder in DynaMechs.cpp file.

Additionally, the TerrainData dialog box GUI is updated for this implementation and the Wireframe overlay function, which will be explained later. The new dialog box is shown before in Figure 2.3. The following files are then also revised to support the implementation of the separate grid resolution: ParserTerrain.h/.cpp, GenerateTerrainFile.cpp, dmvTerrainData.h/.cpp, DlgEnvTerrainData.h/.cpp, and WTKGraphics.cpp.

Similar to the new .cfg file, a new terrain file (.dat) version is needed that contains both X and Y grid resolutions, and also the Wireframe overlay parameters. Unlike the .cfg file, the old version (Version 1) of the .dat file does not have a 'Version' field. Therefore, the new parser must check for the existence of a 'Version' field. If it

does not exist, then it assumes that the terrain file is version 1. Otherwise, it takes the version number specified in the field.

3.3 Wireframe Overlay

The wireframe overlay on terrain feature involves the same files revisions as the X and Y grid resolution feature implementation. A new WTK universe action function that draws the three dimension lines for this, *WTKFGDrawTerrainGrid*, is written under the *CWTKGraphics* class. This function is called every time a new terrain file is loaded, or the user updates the Wireframe overlay parameter in the Terrain Data dialog box.

3.4 Startup Picture

WTK has the capability to load a jpg image file into one of its window using the function *WTwindow_loadimage*. This function loads the indicated jpg image file into the specified window and stretches it so that it fills the window. From debugging the program when using this function, it was found out that this function only takes jpg image file input instead of the bitmap image file written in the manual. The function should only be called once initially when the WTK window is initialized. Also it has to be called inside a WTK universe action function or from a user-specified draw function or it will have no effect. Furthermore, it is called inside the function that draws the axes, *CWTKGraphics::WTKFGDrawCallback*, right after the axes are drawn. Creating a separate WTK universe action function for the startup logo caused the axes to disappear or overlaid on top of the picture, depending on which function is called first in WndWTK.cpp. Therefore, it is important to write the function call in the correct place and point in time.

The drawback in using this function is that someone can find the image file and change it as they wish since the function takes a regular filename input. An idea to prevent anyone changing the image displayed is to write an encryption-decryption code. Another option would be to somehow convert the image file into binary bits and make it an integral part of the RobotBuilder.exe file.

Unfortunately, this feature could not be completely debugged. It seems that the function is called every time the program refreshes the window. Thus, this feature is excluded in RobotBuilder version 1.1. Hopefully, it can be included in the next revision to RobotBuilder. The code for this feature is left inside WTKGraphics.cpp so that future programmer may use it as a reference.

CHAPTER 4

SUMMARY AND FUTURE WORK

4.1 Summary of Work

The view angle implemented in this research allows the user to control the amount of perspective effects on the scene. Another added flexibility to the user is a separate X and Y resolution for the terrain. This feature mainly facilitates the use of RobotBuilder on lower-end computers. An additional feature is the triangle wireframe overlay on the terrain in which the color, the width, and the elevation offset from the terrain can be chosen and saved in the terrain file.

4.2 Future Work

As mentioned earlier, a startup logo for RobotBuilder was not successfully implemented. It would be really nice if a secure startup logo can be included in future revisions of the software. By secure, the startup image should not be easily changed by RobotBuilder user. It has to be protected somehow.

Currently, RobotBuilder is not supported using Microsoft Visual Studio C++ 7.0 / .NET. Compiling or creating the control.dll solutions for the RobotBuilder program can only be done with the older version, Visual Studio C++ 6.0. Support for developing and creating controller for RobotBuilder models using Microsoft Visual Studio .NET would be beneficial.

APPENDIX A

MODIFIED CONFIGURATION (.CFG) FILE FORMAT

In order to preserve the user defined view angle, a new version of the .cfg is needed. This would be Version 3 since Luke Frankart used Version 2. A new version of the configuration file allows the parser to be compatible with previous versions, as it can run the specific codes for the version. The file format below is taken directly and modified from Luke's Version 2 .cfg format [5]. Additions or changes to the Version 2 .cfg format appear in **bold** type.

```
<cfg_file> ::=
{<comment>}
Version 3 [<comment>]
// Configuration file version identifier
{<comment>}
Control_Step_Size <fp> [<comment>]
{<comment>}
Simulation_Display_Rate <int> [<comment>]
{<comment>}
Real-time (TRUE | FALSE) [<comment>]
{<comment>}
Inertial_Axes <fp> <int> [<comment>]
// Axes length and width
{<comment>}
Link_Axes <fp> <int> [<comment>]
// Axes length and width
{<comment>}
Background_Color <int> <int> <int> [<comment>]
// Ints give RGB components and are 0 to 255 inclusive
```

```
{<comment>}
(<euler_integrator> |
<placement_integrator> |
<runge-kutta_fourth_integrator> |
<runge-kutta_adaptive_integrator>)
{<comment>}
Inertial_Axes_As_COI (TRUE | FALSE) [<comment>]
{<comment>}
Stabilize_Vertical_Camera_Position (TRUE | FALSE)
[<comment>]
{<comment>}
COI_Name (<link_in_quotes> | "Inertial Axes") [<comment>]
// "Inertial Axes" is specified when
// Inertial_Axes_As_COI field above is TRUE
{<comment>}
COI_Offset <fp> <fp> <fp> [<comment>]
// The center of interest offset in local coordinates
{<comment>}
Camera_Orientation <fp> <fp> <fp> <fp> [<comment>]
// The camera orientation quaternion
{<comment>}
Camera_Position <fp> <fp> <fp> [<comment>]
// The camera position coordinates in world coordinates
{<comment>}
View_Angle_Degrees <fp> [<comment>]
// The view angle in degrees
{<comment>}
Environment_Parameter_File <file_in_dbl_quotes> [<comment>]
{<comment>}
Control_Dll_File <file_in_dbl_quotes> [<comment>]
{<comment>}
Articulation_File <file_in_dble_quotes> [<comment>]
{<comment>}

<euler_integrator> ::=
Integrator EULER [<comment>]
{<comment>}
Integrator_Parameter_Step_Size <fp> [<comment>]
{<comment>}

<runge-kutta_fourth_integrator> ::=
Integrator RUNGE-KUTTA4 [<comment>]
{<comment>}
Integrator_Parameter_Step_Size <fp> [<comment>]
{<comment>}

<runge-kutta_adaptive_integrator> ::=
```

```
Integrator ADAPTIVE_4_5_RUNGE-KUTTA [<comment>]
{<comment>}
Integrator_Parameter_Epsilon <fp> [<comment>]
{<comment>}
Integrator_Parameter_Minimum_Step_Size <fp> [<comment>]
{<comment>}

<placement_integrator> ::=
Integrator PLACEMENT [<comment>]
{<comment>}

<fp> ::= <floating_point_number>

<comment> ::= #<string_of_characters>
```

APPENDIX B

MODIFIED TERRAIN (.DAT) FILE FORMAT

In order to maintain the X and Y grid resolutions and the wireframe overlay options, a new terrain file version was needed. Unlike the .cfg file, the old version of terrain file does not have a 'Version' field. Thus, the new parser for the .dat terrain file has to check the presence of a 'Version' field. If it does not exist, then the parser assumes that the terrain file is version 1. Otherwise, it takes in the version specified by the 'Version' field. The terrain file format below is taken directly and modified from Steven's Version 1 .dat format [1]. Additions and changes to the Version 1 .dat format appear in **bold** type.

```
<terrain_data_file> ::=
{<comment>}
Version 3 [<comment>]
// Configuration file version identifier
{<comment>}
<x_dim> <y_dim> <x_scale> <y_scale> [<comment>]
<depth_matrix>
{<comment>}
Covering <covering_data>
Wireframe <wireframe_data>

<comment> ::= #<string_of_characters>

<x_dim> ::= <positive_integer>
// count in x direction

<y_dim> ::= <positive_integer>
```

```
// count in y direction

<x_scale> ::= <positive_floating_point>

<y_scale> ::= <positive_floating_point>

<depth_matrix> ::= {<depth_row>} [<comment>]

<depth_row> ::= <fp> <fp> {<fp>}
// the depth matrix is x_dim by y_dim in size

<fp> ::= <floating_point_number>

<covering_data> ::=
(<simple_color> | <texture>)

<simple_color> ::=
Covering: COLOR <int> <int> <int> [<comment>]
// Ints give RGB components and are 0 to 255 inclusive
{<comment>}

<texture> ::=
Covering: TEXTURE <path_in_dbl_quotes> [<comment>]
     {<comment>}

<wireframe_data> ::=
(ON | OFF) <color> <line_width> <z_offset>

<color> ::= <int> <int> <int>
// Ints give RGB components and are 0 to 255 inclusive

<line_width> ::= <pos_fp>

<z_offset> ::= <fp>

<pos_fp> ::= <positive_floating_point_number>
```

BIBLIOGRAPHY

[1] S. Rodenbaugh, "RobotBuilder: A Graphical Software Tool for the Rapid Development of Robotic Dynamic Simulations," M.S. Thesis, Department of Electrical Engineering, The Ohio State University, December 2002.

[2] Scott McMillan, *Computational Dynamics for Robotic Systems on Land and Under Water*, PhD dissertation, The Ohio State University, Columbus, OH, 1994.

[3] Scott McMillan, David E. Orin, and Robert B. McGhee, "Object-Oriented Design of a Dynamic Simulation for Underwater Robotic Vehicles," *IEEE International Conference on Robotics and Automation*, pp. 1886-1893, 1995.

[4] *WorldToolKit Reference Manual Release 9*, software reference manual, Engineering Animation, Inc., April 1999.

[5] Luke Frankart, "Implementation of Additional Features in RobotBuilder to Increase Its Capability to Simulate Legged Robots" B.S. Honors Thesis, Department of Electrical Engineering, The Ohio State University, 2003.

[6] Min-Hsiung Hung, "Dynamic control and simulation of actively-coordinated robotic terrain-adaptive wheeled vehicles" Ph.D. Thesis, Department of Electrical Engineering, The Ohio State University, 1999.