RobotBuilder

Example System Tutorial:

The Simplified Whegs Robot Model

Lucas Frankart
David E. Orin
Department of Electrical Engineering
The Ohio State University

June 5, 2003

Background

RobotBuilder and RobotModeler were originally developed as part of Steven Rodenbaugh's Master's thesis work. Further work was conducted by Lucas Frankart for his Bachelor's thesis. For more information about the applications, please consult the User's Guides or the RobotBuilder website at http://eewww.eng.ohio-state.edu/~orin/RobotBuilder/RobotBuilder.html.

TABLE OF CONTENTS

Background ii

1. Introduction 1

2. Background 1

3. Designing the Links in RobotModeler 3

4. Constructing the Robot in RobotBuilder 9

5. The Environment 12

6. Setting the User View 14

7. The Controller 15

8. Simulation 20

9. Whegs Simulation Files 21

10. Conclusion 24

Bibliography 25

1. Introduction

RobotBuilder and RobotModeler are two applications developed by Steven Rodenbaugh for the rapid development of robotic simulations. This document serves as an introductory tutorial to both software packages through the development of a simplified model of the Whegs robot [1]. Although every effort was made to make this document as accessible as possible to a new user of RobotBuilder and RobotModeler, some assumptions have been made. First, it is assumed that the reader is familiar with the Microsoft Windows operating system and the associated user interface (i.e. the use of menus, dialog boxes, etc.). Secondly, it is assumed that the user has the latest version of RobotBuilder and RobotModeler installed and working on his/her computer (Version 1.0 as of the writing of this tutorial). Finally, it is assumed that the user has access to an installation of Microsoft Visual C++ and has some familiarity with the C/C++ programming languages. This knowledge is needed to develop and compile the controller for the model.

The model developed in this tutorial is distributed with the RobotBuilder package. In several sections, some detail was omitted. In these areas, the user may consult the existing model for further assistance. Note that thorough User's Guides are also distributed with RobotBuilder and RobotModeler for further assistance.

2. Background

Whegs is a legged hexapod robot developed under the direction of Dr. Roger Quinn as one of many biologically-inspired robots in the biorobotics program at Case

Western Reserve University [1]. Although the inherently simple design made it straightforward to implement, several unique features employed by Whegs made it an interesting example for simulation. These included the design of the wheel-leg ("wheg") and the power train.

Whegs achieves locomotion through the use of six wheel-legs ("whegs"). Each wheg consists of three spokes spaced at 120° about a central revolute joint. The wheg acts as a three-spoke wheel without a surrounding rim. This configuration achieves a few unique advantages. First, generally only three points per wheg contact the terrain, as opposed to the continuous contact a wheel would have. This permits Whegs to more easily cross rough and varied terrain. Secondly, the omission of a rim on the wheel allows Whegs to approach and overcome obstacles higher than would be possible using a wheel of similar radius. Figure 2.1 illustrates this advantage.

Figure 2.1: Demonstration of Whegs Ability to Overcome Taller Obstacles than Wheels of Similar Radius [1]

The second unique feature of the Whegs robot is the inclusion of passive compliance in each of the wheg revolute joints. Over level terrain, each wheg is

nominally 60° out of phase with the wheg directly across from it. The front and back whegs on one side are in phase with the middle wheg on the other side. In this manner, the robot moves with a tripod gait over level terrain. Upon encountering an obstacle, torsional springs in each revolute joint allow the whegs to rotate into phase, thereby applying force from both whegs to the surface of the obstacle. In this manner, Whegs is able to overcome the obstacle, and then return to a tripod gait. Figure 2.2 illustrates this concept. Forces from the obstacle onto each wheg cause them to rotate until they are in phase.

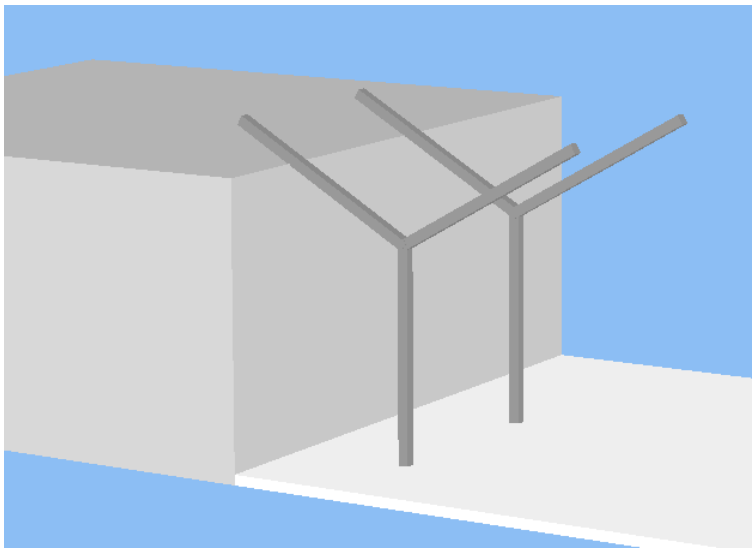

Figure 2.2: Using Passive Compliance to Allow Whegs to Rotate Into Phase [1]

3. Designing the Links in RobotModeler

The first step in designing any robot for use in RobotBuilder is to design the graphical model to be used for each of the links. Two unique links are used in the Whegs robot: the wheg itself, and the body of the robot. RobotModeler provides an easy

interface for generating graphical models based on a combination of primitives such as cubes (block primitives), spheres, cones, cylinders, hemispheres, and truncated cones. Each primitive is added individually to the model, and then scaled, rotated, and translated appropriately to form a portion of the link. By defining the mass of each primitive as it is added, RobotModeler will calculate the composite inertial properties of the link for use in RobotBuilder.

Specifics of the physical dimensions and mass of Whegs were needed in order to develop a simulation model. Table 3.1 summarizes the dimensions and mass of the Whegs I robot. Because individual masses for each component were not available, some assumptions were made in developing the model. Each of the six whegs was assumed to represent 0.05 kg of the total mass. The remaining 2.6 kg of mass was evenly distributed throughout the framework of the body. Details of the drive train were not available so it was not included in the simulation model.

Mass (overall)	2.9 kg
Length (front to back)	51.4 cm
Width (leg to leg)	51.4 cm
Wheg Spoke Length	10.2 cm

Table 3.1: Whegs I Physical Dimensions and Mass

Figure 3.1 shows the wheg link as it was created in RobotModeler. The wheg link consists of three block primitives rotated at 120° intervals about the link coordinate system. Table 3.2 contains the dimensions, mass, translation and quaternion of each of

the three block primitives. (One note should be mentioned concerning the use of units in both RobotBuilder and RobotModeler. Neither program explicitly associates units with any of the measurements used. Rather, it is up to the user to maintain consistent units. All units were converted to the MKS system for entry in both RobotBuilder and RobotModeler.) By entering these values in the properties sheet of each primitive (accessed by right-clicking the primitive and selecting "Properties…"), the wheg link may be manually reconstructed.

In practice, however, a combination of manual entry of dimensions and use of the GUI controls is used. For this link, the dimensions and mass of each primitive were manually entered in the properties sheet. The dimensions and mass listed were approximated from the actual Whegs I robot. The mass of each primitive was selected such that the total mass of the wheg was approximately 0.05 kg. Then, each block primitive was moved to the appropriate location by selecting "Rotate (degrees)" as the adjustment type, setting the step size to 60, and clicking the Z+ (Z-) button to rotate the link 60° about the Z axis of the link frame. Then, each primitive was translated half the spoke length by selecting "translate" as the adjustment type, entering a step size of 0.051, selecting "Adjust With Respect To: Local Frame", and clicking the X- (X+) button.

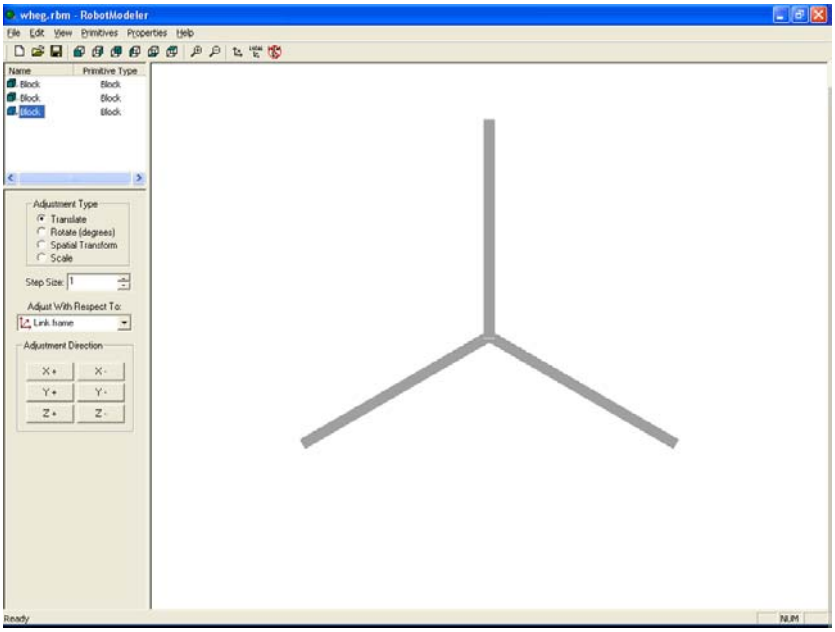

Figure 3.1: Wheg Link in RobotModeler

Primitive	X Length (m)	Y Length (m)	Z Length (m)	Translation (m)	Quaternion	Mass (kg)
Spoke 1	0.102	0.005	0.005	[-0.025, 0.0433013, 0]	[0, 0, -0.866025, 0.5]	0.0166667
Spoke 2	0.102	0.005	0.005	[-0.025, -0.0433013, 0]	[0, 0, 0.866025, 0.5]	0.0166667
Spoke 3	0.102	0.005	0.005	[0.05, 0, 0]	[0, 0, 0, 1]	0.0166667

Table 3.2: Dimensions, Mass, Translation and Quaternion of Primitives Used to Form Wheg Link

As the wheg graphical model is the first of a number of files needed for this simulation, a note should be included concerning file and directory structure. A simulation is typically comprised of a number of different files. Section 9 details all of the files required for this particular model, and outlines the general directory structure that should be employed.

Figure 3.2 shows the body link as it was created in RobotModeler. The body link was formed by using multiple block primitives of equal cross-sectional area (1 cm x 1 cm) and varying length. The cut/copy/paste functionality implemented in RobotModeler was of great use in the creation of the body. First, the top structure of the link was created from multiple block primitives. These were added in the same manner as the primitives for the wheg, where each primitive's mass and dimensions were entered in the properties sheet, and then the primitive was appropriately rotated and/or translated. Next, all primitives that appear in the top surface were selected and copied, and then one of each duplicate primitive was translated half the body thickness in the +Z direction. The other was translated half the body thickness in the –Z direction. Finally, vertical struts were included to join the two halves of the body.

The mass for each primitive was calculated by dividing the assumed 2.6 kg total mass of the body structure by the total length of cubic primitives. This value, when multiplied by the individual primitive length, resulted in a uniform distribution of the mass throughout the structure of the body. Note that the axles were assumed to contribute negligibly to the overall mass of the body, and were only included for enhanced graphical detail.

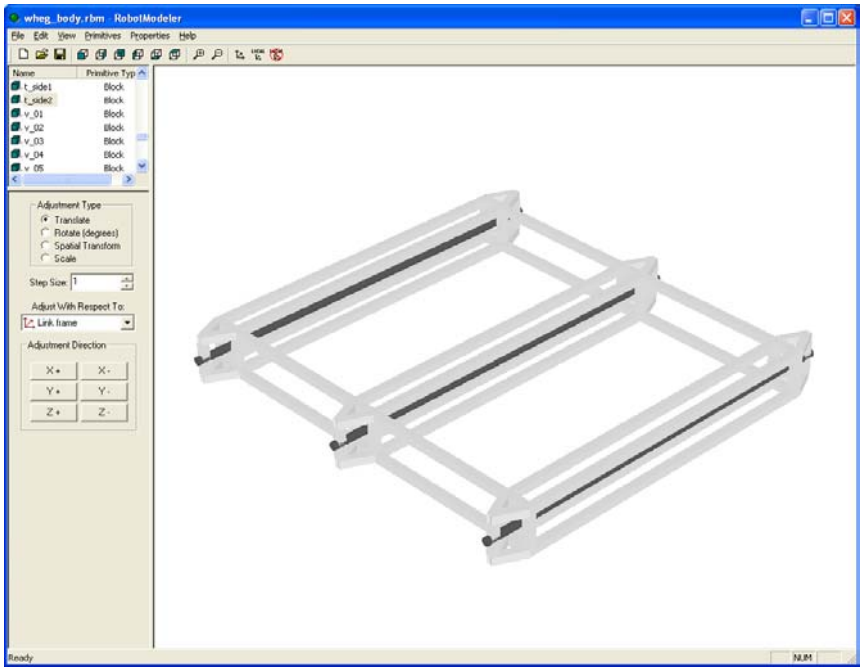

Figure 3.2: Body Link in RobotModeler

As noted above, RobotModeler is capable of calculating the composite inertial properties of a link. This information may then be imported into RobotBuilder for use in the simulation. The decision to use this information is a per-link setting that must be set in RobotModeler to maintain consistency. The inertial properties calculated for both the wheg and body links by RobotModeler were used in the simulation. Table 3.3 contains the center of gravity and inertia tensor matrix for each link.

Link	Center of Gravity	Inertia Tensor ($kg \cdot m^2$)		
Body	[0, 0, 0]	7.37396e-008	0	0
		0	5.29125e-008	0
		0	0	7.37396e-008
Wheg	[0, 0, 0]	5.25477e-005	2.42842e-005	0
		2.42842e-005	4.524e-005	0
		0	0	5.61515e-005

Table 3.3: Whegs Link Centers of Gravity and Inertia Tensor Matrices

4. Constructing the Robot in RobotBuilder

After the graphical model of each link is complete, it is a simple matter of constructing the robot in RobotBuilder. The left pane of the Build mode in RobotBuilder contains a tree view of the robot currently under construction. This tree view shows a hierarchy of links that illustrates the way multiple links are connected. To begin the model, right-click on "Articulation" and choose "Add Child…", "Static Root Link", the foundation for the robot. A child link of type "Mobile Base Link" is added in a similar fashion for the body. After a link is added, right-clicking the link and choosing "Properties…" will launch a dialog box which allows properties of that particular link to be modified.

For the body link, the first step is specifying the graphical model. Because the graphical model was designed in RobotModeler, the inertial parameters are filled in automatically. The only other parameters that must be specified for the body are the initial position vector and the contact points. The initial position vector may place the robot anywhere in RobotBuilder space. Because of the way RobotBuilder calculates

contact forces, the robot is typically placed initially suspended above the terrain (positive Z_I direction), such that it will fall to the surface upon simulation. If the robot were placed such that a portion of one or more links extended below the terrain surface, a potentially large spring force could result that would cause the robot to fly up into the air.

Contact points are the points on a given link at which forces will be generated upon interaction with the terrain surface. In an effort to account for the body becoming caught on protrusions in the terrain, several points were specified along the underside of the body link. The number and location of these points is somewhat arbitrary, and depend on the type of terrain the robot will be crossing, the importance of accurately simulating the interaction of the body with the terrain, and the amount of computational complexity one is willing to accept. For this model, three contact points were placed on each of the three cross members of the body: one at each end, and one in the middle.

Each of the six wheg links are connected with *revolute joints.* They are added as children of the body just as the above links were added. Note that this is an example where the naming of each link-joint pair is crucial to the correct operation of the robot. As discussed below, the control program specifies each link by name. This implies that each link name should be unique, and that the link names should be consistent with those expected by the controller. In this example, the link names must be consistent with the controller to ensure proper phasing between the whegs. Figure 4.1 is a simplified overhead view of the model which illustrates both the naming convention for the wheg links assumed by the controller, and the relationships between the local link coordinate systems.

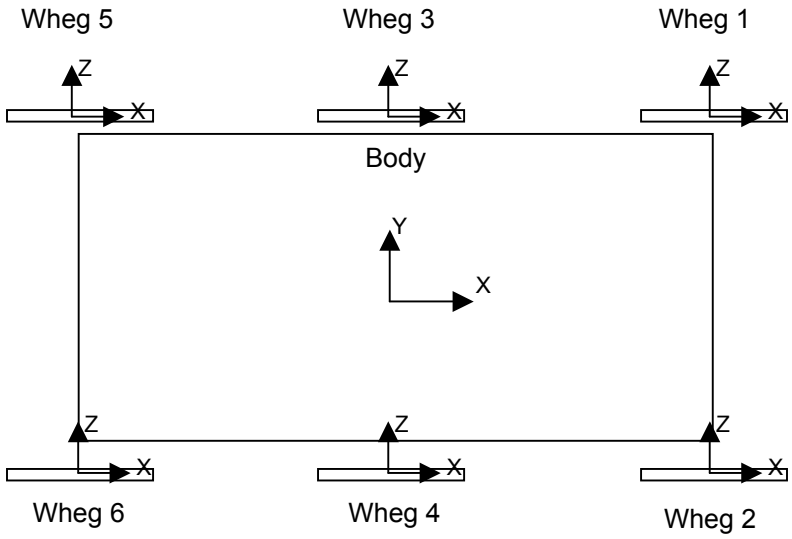

Figure 4.1: Simplified Overhead View of the Whegs Model to Illustrate Naming Convention and Coordinate System Relationships

After each wheg link is added and named, the properties must be set. As above, choosing the RobotModeler graphical model file will automatically configure the link inertial parameters. Next, the orientation of each wheg must be specified relative to the parent link, the body, using Modified Denavit-Hartenberg (MDH) parameters [2]. In addition to orienting the whegs appropriately, the MDH Parameters for the whegs also configure the initial phasing of the whegs. Each wheg is rotated about the body forward axis, translated to its appropriate location, and then rotated about the axle to the appropriate phasing. For revolute joints, the parameter θ specifies the rotation of the joint. By setting the initial θ of horizontally opposed whegs to either 0 or $\pi/3$, the whegs can be initially configured 60° out of phase and to operate in a tripod gait. The MDH parameters for each wheg are given in Table 4.1. Please consult Figure 4.1 for the relationship

between the wheg link and body link coordinate systems. Finally, contact points must be added at the end of each of the three spokes on each wheg.

Link Name	a	α	d	θ
Wheg 1	0.23	-1.5708	0.25	0
Wheg 2	0.23	-1.5708	-0.25	1.0472
Wheg 3	0	-1.5708	0.25	1.0472
Wheg 4	0	-1.5708	-0.25	0
Wheg 5	-0.23	-1.5708	0.25	0
Wheg 6	-0.23	-1.5708	-0.25	1.0472

Table 4.1: Modified Denavit-Hartenberg Parameters for the Wheg Links

5. The Environment

After a simulation model is constructed, the environment must be specified. The environment properties encompass the terrain topology and characteristics as well as the gravity vector.

Terrain in RobotBuilder is specified as an M x N matrix of elevation values (positive Z_I direction of *DynaMechs* inertial frame). For the Whegs simulation, it was desirable to evaluate the robot's ability to overcome obstacles. As such, a terrain was specified which contained multiple abrupt changes in elevation. Although true step discontinuities are not possible in RobotBuilder, they may be approximated by choosing a sufficiently fine resolution (sufficiently large M and/or N values). In this manner, the multi-stepped terrain utilized with the Whegs simulation was created. Because increasing the number of terrain points specified increases the computational complexity of the simulation, a compromise was made. The terrain was designed to be long and narrow

such that the number of points specified could be reduced, yet still produce a good approximation to a step discontinuity in the direction of motion.

The terrain was created by manually creating the elevation matrix in a spreadsheet. Because the terrain is prismatic in nature (constant cross section), the elevation data for the first column of the matrix was manually entered to define the multi-stepped terrain. This column was then replicated the appropriate number of times to generate the overall terrain width. Finally, this information was copied into a text file, and the few additional bits of information to form a complete terrain definition file were added. For more details, please see the `step.dat` file included with the distribution.

With the terrain topology specified, next comes the matter of defining the terrain characteristics. Ground normal and planar spring and damper constants are used to define how the robot will interact with the surface. The values chosen for Whegs are typical values for a hard terrain, and were taken from the Quadruped example distributed with the RobotBuilder package. The gravity value chosen was -9.81 m/s^2 (in the $-Z_I$ downward direction), earth's gravity at sea level. Figure 5.1 shows the completed Whegs robot model with the terrain in RobotBuilder.

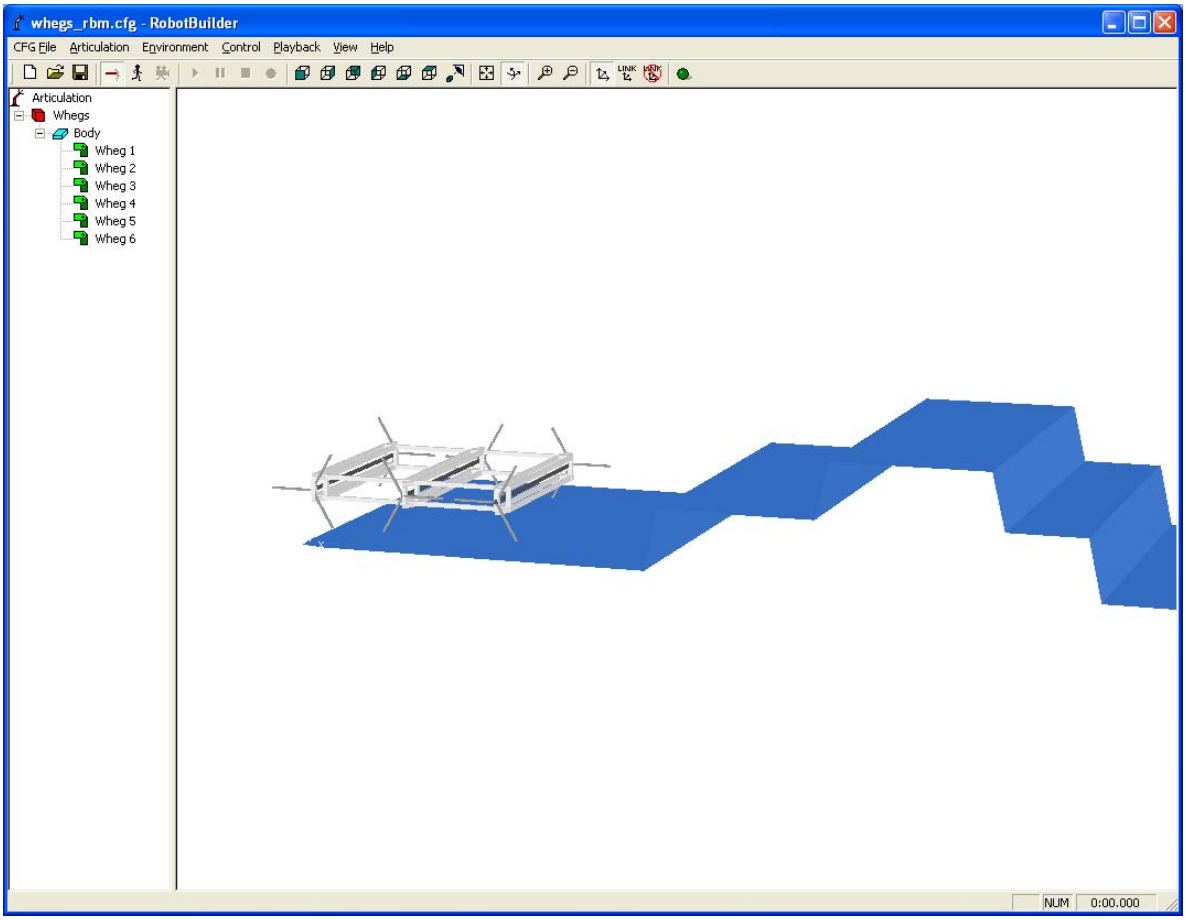

Figure 5.1: The Completed Whegs Robot Model in RobotBuilder

6. Setting the User View

Now that the model has been built and the terrain specified, it is a good time to set the user-defined view. The first step is to set a center of interest. The center of interest is the link in the model (or optionally the inertial frame) that the camera is directed at. For the Whegs simulation, the body is selected as the center of interest by right clicking with the mouse on "Body" in the tree view (the left pane), and selecting "Center of Interest".

Next, using the mouse in the main window, orient the view so that it is in a pleasing position. Note that there are two modes of camera control: Pan and Examine.

When the camera is in Examine mode, left mouse clicks around the model will rotate the camera. In Pan mode, left-clicks of the mouse will translate the viewport window. In both modes of camera operation, right-clicks of the mouse in the main window will zoom the camera in or out.

After a suitable view is obtained, set the user view by opening the Simulation Properties dialog box (located on the "CFG File" menu). Select the "Camera" tab, and click on the button marked "Set". This will store the current view. Note that the user view information is stored in the configuration file, so now is a good time to save this file. After the user view has been set and saved as part of the configuration, it becomes the default view each time the simulation is opened.

7. The Controller

The controller is probably the most crucial part of the entire simulation. It defines not only how the robot will behave during the course of simulation, but also the user interface for controlling the robot's behavior. Great flexibility is afforded by allowing the user to define the controller in a C/C++ program. This flexibility was used to full advantage in implementing the unique mechanics of the Whegs robot.

The user-designed controller consists of four main functions. The first is an initialization function called by RobotBuilder upon initiation of a simulation. The initialization function sets up pointers to the links to be controlled in the model, defines the user interface elements, and performs any other controller-specific initialization. The next function is the controller itself. This function is called at every control step interval during simulation (defined in the simulation configuration file). This function retrieves

the state variables from the *DynaMechs* engine, and calculates new torques to be applied to all joints. The final two functions are an uninitialize routine called upon termination of a simulation to clean up any user-defined data structures, and a function to handle key presses during simulation.

In the Whegs initialization function, the first step is to retrieve pointers to each of the six whegs links. This is where the uniqueness of link names becomes important. A pointer to each link is retrieved by calling a RobotBuilder function that allows a user to specify a link by its name.

After these six pointers are retrieved, the user interface controls are defined. During simulation, the user has control over the motor voltage, gear ratio, and wheg drive spring and damper constants. User control of these simulation parameters is provided through the use of slider controls. This allows the user to experiment with different drive train parameters. In addition, the control outputs the individual wheg positions (in radians) at one second intervals to a list box. Figure 7.1 shows the user interface implemented by the Whegs controller. Because the slider controls return only integer values, a scaling factor is applied to the spring and damper constants. For the spring constant, the value selected on the slider is actually 10 times the value used in the simulation (e.g. a slider value of 58 corresponds to 5.8 in the simulation). Similarly, the value on the damper constant slider is divided by 1000 before being used in the simulation.

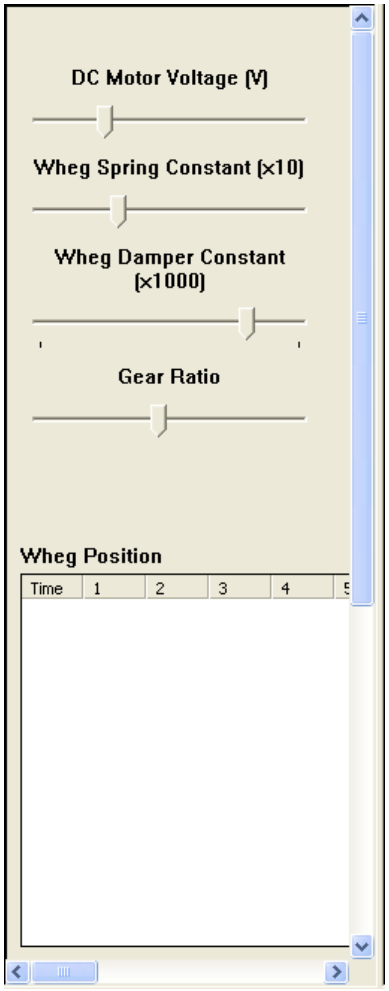

Figure 7.1: Whegs Controller User Interface

Next, the actual control function is specified. The passive compliance in each wheg that is fundamental to Whegs' operation was implemented by first simulating the motor used to supply torque to the whegs. A standard 48 V DC motor was chosen to power the Whegs simulation [3]. The motor parameters of the particular motor chosen are listed in Table 7.1. Note that the actual motor and drive train of Whegs I are not modeled here. However, the main characteristics are simulated.

Voltage	0-48 V DC
K_τ	60.3 mN·m/A
K_B	60.3 mV/(rad/sec)
R_m	1.16 Ω
J_m	134 g·cm^2
B_m	0.0

Table 7.1: DC Motor Parameters

This motor is connected through a drive train, with a characteristic gear ratio, to each of the six whegs. At each control step (when the control procedure is called), the total torque applied to each wheg is calculated. Based on this information and the previous motor shaft rate and position, a new acceleration, rate, and position are calculated. The individual wheg torques are passed back to RobotBuilder, and the new motor state variables are preserved locally for the next iteration.

Equation 7.1 shows the calculation performed for each wheg to determine the torque applied during that control step.

$$\tau_i = k\left(\frac{\theta_m}{n} - \theta_i + \varphi_i\right) + b\left(\frac{\omega_m}{n} - \omega_i\right) \qquad (7.1)$$

In this equation, τ_i is the torque applied to the i-th wheg, k is the torsional spring constant, θ_m is the current motor shaft position, n is the gear ratio of the drive train through which the whegs are coupled, θ_i is the current position of the i-th wheg, φ_i is the set phase of the i-th wheg (0 or 60°), b is the torsional damper constant, ω_m is the current motor shaft speed, and ω_i is the current speed of the i-th wheg. After these torques are calculated, they are passed back to *DynaMechs* to be applied to the model for the next simulation interval.

The equations below show the calculation for each of the motor state variables.

$$\dot{\omega} = \frac{1}{J_m}\left[K_\tau \left(\frac{V_m - K_B \omega}{R_m} \right) - B_m \omega - \frac{1}{n} \sum \tau_i \right] \tag{7.2}$$

$$\theta \leftarrow \omega \cdot \Delta t + \theta \tag{7.3}$$

$$\omega \leftarrow \dot{\omega} \cdot \Delta t + \omega \tag{7.4}$$

In these equations, $\dot{\omega}$, ω, and θ are the motor shaft acceleration, rate, and position, respectively; J_m is the motor inertia; K_τ is motor torque constant; V_m is the motor supply voltage; K_B is the back-emf constant; R_m is the motor armature resistance; B_m is the motor damping constant; n is the gear ratio of the drive through which the whegs are coupled; τ_i is the torque of the i-th wheg; and Δt is the amount of simulation time that has elapsed since the control program was last called (the control step). Note that the state variables are calculated in the order listed above so that the acceleration and position are based upon the motor shaft rate calculated during the previous control step. These state variables are maintained locally in the control function and are updated each iteration through this simple Euler integration scheme.

In addition to the wheg position list output during simulation, the controller also writes a data file with more detailed information of the simulation. At approximately 0.1 second intervals, the data file is updated with current simulation data. First the simulation time is output, followed by the voltage, spring and damper constants, and gear ratio at that simulation time. Next, position information is output for each wheg. Both relative wheg position (i.e. the position of the wheg relative to the drive train position: $\theta_m/n - \varphi_i$) as well as absolute (the actual rotation) position data is output. Finally the rate and torque

of each wheg is written to the file. The file is created in comma-separated values format (`.csv`) for use in most spreadsheet programs (such as Microsoft Excel).

The source code for the Whegs controller may be found in the `RobotBuilder\Projects\Whegs\Control` directory. It was based upon the example control program (Skeleton Control) distributed with the RobotBuilder package.

8. Simulation

The final parameters to set before actually running a simulation are the simulation properties. These parameters are set in the `Simulation Properties` dialog box (located on the "CFG File" menu). On the first tab that opens ("Simulation"), begin by selecting an integrator. "Runge-Kutta (4th Order)" was selected for the Whegs simulation as it provides a good numerical approximation with acceptable computational overhead. A step size of "0.0001" is appropriate for the Whegs simulation, as values larger than this may cause the controller to become unstable. If "softer" ground is used (by specifying different environment properties), a larger value for the integrator step size may be appropriate.

In the "Control Step Size" edit box, a value of 0.0005 was chosen. This value is typically larger than the value used for the integrator step size, but the actual values to be used depend upon the poles of the system dynamics. Next, the check box beside "Slow simulation to real-time" is checked. Although most computers are not able to simulate a model faster than real-time, enabling this check box ensures the simulation will not move faster than expected. The final option found here, "Display Update Period", specifies how often the display is updated in terms of control steps. If the simulation is run on a

computer with a reasonably good graphics card, this value may be left at one. For slower graphics cards, a larger value may be appropriate to speed up the simulation.

After selecting the control DLL (accessed under the "Control" menu), it is possible to actually run a simulation. To run a simulation, simply switch from `Build` mode to `Simulate` mode by selecting the toolbar button with an icon of a running person (). This will load the control program, and enable the "Play" (), "Stop" (), "Pause" (), and "Record" () buttons on the toolbar. Pressing "Play" will begin simulation of the model. The simulation may be recorded by selecting "Record" before or during simulation. Switching to `Playback` mode (by selecting the button with a video camera icon: ) will allow playback of a previously recorded simulation.

9. Whegs Simulation Files

As noted above, the development of a typical simulation in RobotBuilder involves a number of different files. The number of files required and the significance of each is not always clear to the new user. Table 9.1 lists all of the files required for the Whegs model, with a brief description of each.

File Name	Description
`Control.dll`	The user-defined Dynamic Link Library that implements the controller for the simulation.
`step.dat`	The terrain data file. This file contains elevation data for the terrain, as well as the covering color.
`wheg.rbm`	The graphical model for the wheg link. This file, produced by RobotModeler, also contains the physical properties of the link (inertia tensor, mass, and center of gravity).
`whegs_body.rbm`	The graphical model for the body link. This file, produced by RobotModeler, also contains the physical properties of the link (inertia tensor, mass, and center of gravity).
`whegdata.csv`	Simulation data produced by the example controller that contains information about the last simulation.
`whegs.cfg`	The configuration file. This file contains all of the simulation options (integrator type, control step size, etc.), user preferences (link axes size, background color, user camera view, etc.), and the filenames of the articulation, control, and environment files.
`whegs.dm`	The articulation file. This file contains all information needed to define the relationship between all of the links in the model.
`whegs.env`	The environment file. This file contains all of the terrain characteristics (ground planar and normal spring and damper constants, surface friction), the gravity vector, and the filename of the terrain data file that specifies terrain elevations.

Table 9.1: Files Required for the Whegs Simulation

Setting up the appropriate directory structure is an important step when creating a new simulation. All of the simulation files described above are typically placed in a new directory under the "Projects" directory in the main RobotBuilder directory. The Whegs

simulation, for example, is located in the `RobotBuilder\Projects\Whegs` directory. Although the simulation will open and operate correctly as long as all of the above files are located in the same directory anywhere in the directory structure, this scheme is recommended, and adherence to this structure facilitates development of the control DLL.

To begin development of a controller, the entire `Control` directory from the `RobotBuilder\Projects\Skeleton Control` is first copied to the newly created project directory (`RobotBuilder\Projects\Whegs`, in this example). *Microsoft Visual C++* is used to develop the controller; if the recommended directory structure was used, the provided *Microsoft Visual C++* workspace file will be configured appropriately for all dependencies (the control DLL depends on several libraries distributed with RobotBuilder). It is then only a matter of writing the C++ code to implement the controller and compiling the DLL. Again assuming the above directory structure is used, the compiled DLL will be placed in the project directory, ready to be selected in RobotBuilder.

When the `.cfg` file is changed during the course of using RobotBuilder, the user will be prompted to save changes before quitting the program. The other files created with RobotBuilder or RobotModeler are not monitored for changes; consequently, it is the user's responsibility to save these files before terminating the program to prevent data loss.

10. Conclusion

Hopefully, this tutorial will be helpful in your research. For further assistance, please consult the RobotBuilder and RobotModeler User's Guides. If you discover any bugs with either program or typographical errors in this document, please send an email to robotbuilderbugs@yahoo.com.

Bibliography

[1] R. D. Quinn, J. T. Offi, D. A. Kingsley, and R. E. Ritzmann, "Improved Mobility through Abstracted Biological Principles," in *Proc. Of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems,* (EPFL, Lausanne, Switzerland), pp. 2652-7, October 2002.

[2] John J. Craig, *Introduction to Robotics: Mechanics and Control, 2nd Edition,* Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[3] *Maxon RE 40 DC Motor Specification Sheet*, Maxon Precision Motors, Inc., April 2002.