

Team 505 Operation Manual

Milton Bouchard, Michael Dina, Onoriode Onokpise, Jackson Raines, and Zachary Shapiro

FAMU-FSU College of Engineering

Author Note

This project is sponsored by the Center for Intelligent Systems, Control, and Robotics (CISCOR)

OPERATION MANUAL	2
Using the Tool.....	5
Acronyms and Terminology.....	5
MATLAB GUI.....	5
Starting a New Project	5
Loading a Previous Project	6
Database Browser	6
Creating a New Database.....	7
Importing a Database	8
Robot Design	9
The Sketch View	10
Scaling Properties.	10
Running Calculations.....	11
Deliverables (Work in Progress).....	12
Bill of Materials	12
Web Resources	13
Architecture and Models.....	15
System Composer	15
Architecture Overview	15
Simulink.....	18
Modeling Assumptions	18
Motor Model.....	20

Battery Model 22

Leg Linkage Model..... 23

Source Code 24

MATLAB GUI Code..... 24

GitHub Access 32

OPERATION MANUAL	4
Figure 1 Design Tool Start Page	5
Figure 2 The Database Browser.....	6
Figure 3 Creating a New Database	7
Figure 4 The Database Creator	7
Figure 5 Loading a Database	8
Figure 6 Robot Design Tab	9
Figure 7 Robot Sketch View	10
Figure 8 Dynamic Scaling	10
Figure 9 Run Simulink Analyses	11
Figure 10 Deliverables Tab	12
Figure 11 Generating a Bill of Materials	12
Figure 12 Saving Values	13
Figure 13 Website links to motor	13
Figure 14 Launching Simulink	15
Figure 15 Simulink Start Page	15
Figure 16 Start a new System Composer Architecture	16
Figure 17 System Composer Architecture	17
Figure 18 Linking to Simulink Models.....	18
Figure 19 Motor Model Architecture	21
Figure 20 To Workspace Components	21
Figure 21 Out Bus Element.....	22
Figure 22 Battery Model Architecture	23
Figure 23 Leg Linkage Model Architecture.....	24

Using the Tool

The following sections explain how to use the design tool from uploading a database to utilizing the output deliverables.

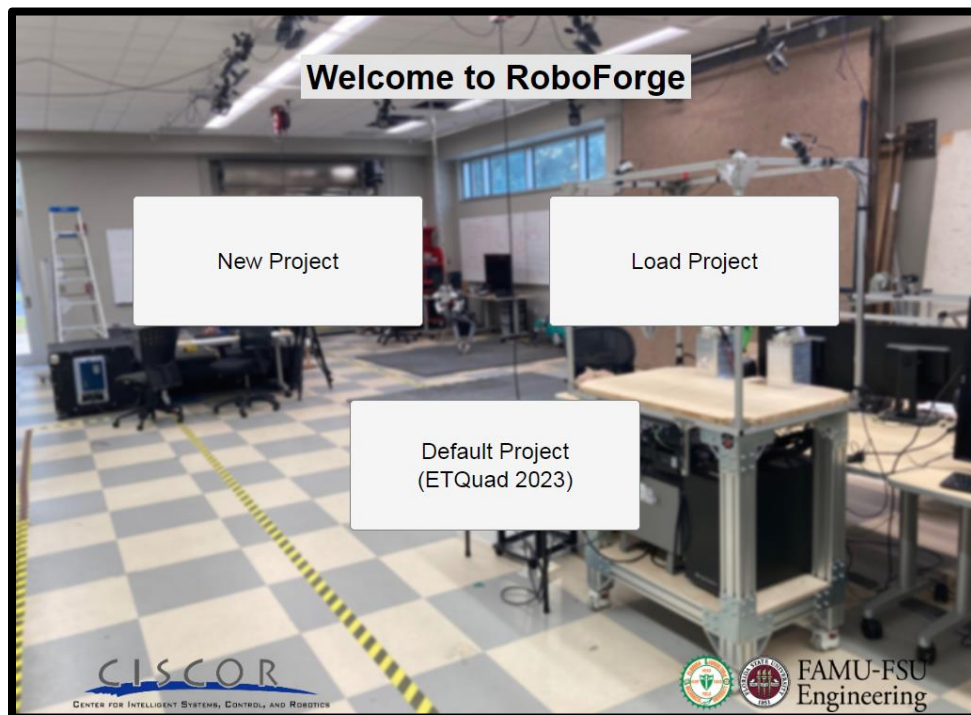
Acronyms and Terminology

MATLAB GUI

Starting a New Project

Figure 1 shows the start page of the design tool. To start a new project, open the tool and select “New Project”. The tool will proceed to the Database Browser tab with basic values in place. You can also start a new project from the File menu.

Figure 1
Design Tool Start Page



Note. When the user starts our tool, they are greeted with this page.

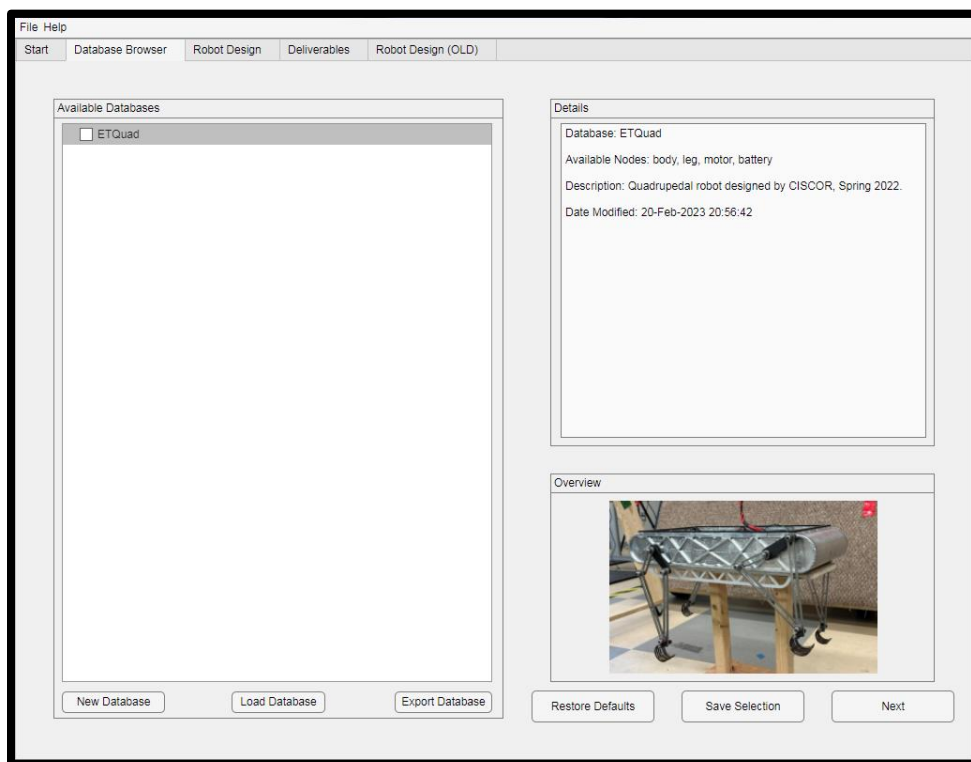
Loading a Previous Project

To load a previous project, select “Load Project” at the start screen. Select an appropriate project “.m” file, and your previous robot databases, adjustment values, and deliverables will be loaded.

Database Browser

Figure 2 shows the Database Browser. The Database Browser allows for selecting which robot database is currently active in the tool. Select a database to view its description and contents in the details pane on the right-hand side.

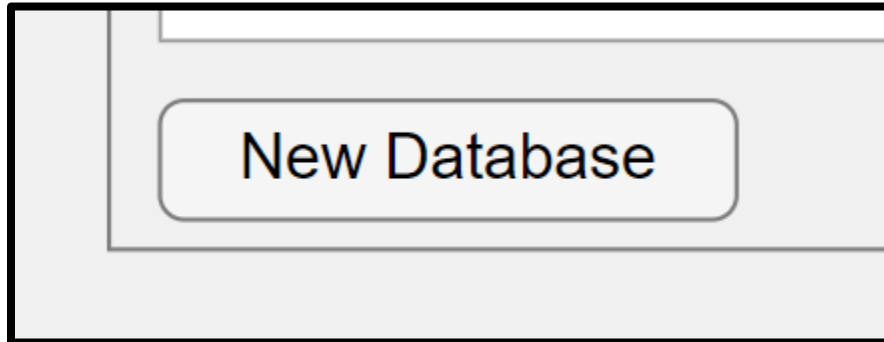
Figure 2
The Database Browser



Note. The Database Browser allows the user to view robots already in their database and add new ones.

Creating a New Database

Figure 3
Creating a New Database



To create a new database, select the “New Database” button in the lower left corner of the Database Browser screen. Figure 3 shows this button. You can also select “New Database” from the File menu. Selecting this will load the Database Creator window.

The Database Creator

The Database Creator allows for the creation of new robot databases, based on data collected from robot measurements. Figure 4 shows this window. The name entered in the Robot Name field will be used as the filename for the final database file. Enter in the physical characteristics or attributes for each field listed, then click Save to File to record this information. The database file format is a “.mat” Matlab file.

Figure 4
The Database Creator

New Robot Database

Please provide the relevant data.

General

Robot Name

Description

Image

Physical Characteristics

Body Length (cm)

Body Width (cm)

Body Height (cm)

Leg Length (cm)

Leg Width (cm)

Leg Thickness (cm)

Total Mass (kg)

Single Leg Mass (kg)

Sensor Array Mass (kg)

Motor Attributes

Motor Mass (kg)

Stall Torque (Nm)

No-load Speed (rpm)

Battery Attributes

Capacity (mAh)

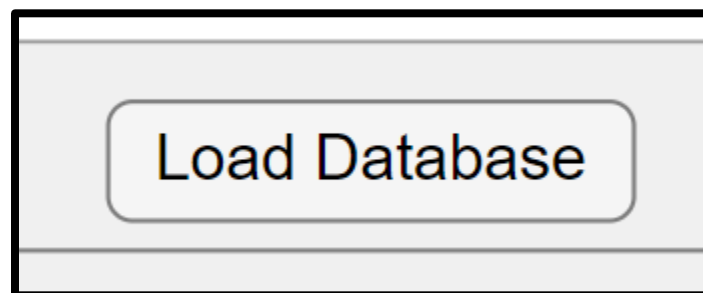
Battery Mass (kg)

Note. The user can create a new robot database by providing the robot’s parameters.

It is greatly recommended to include both a description and an image file for future reference, as these display once the database is selected in the Database Browser. Once finished with your entry, close the Database Creator window to return to the main window.

Importing a Database

Figure 5
Loading a Database



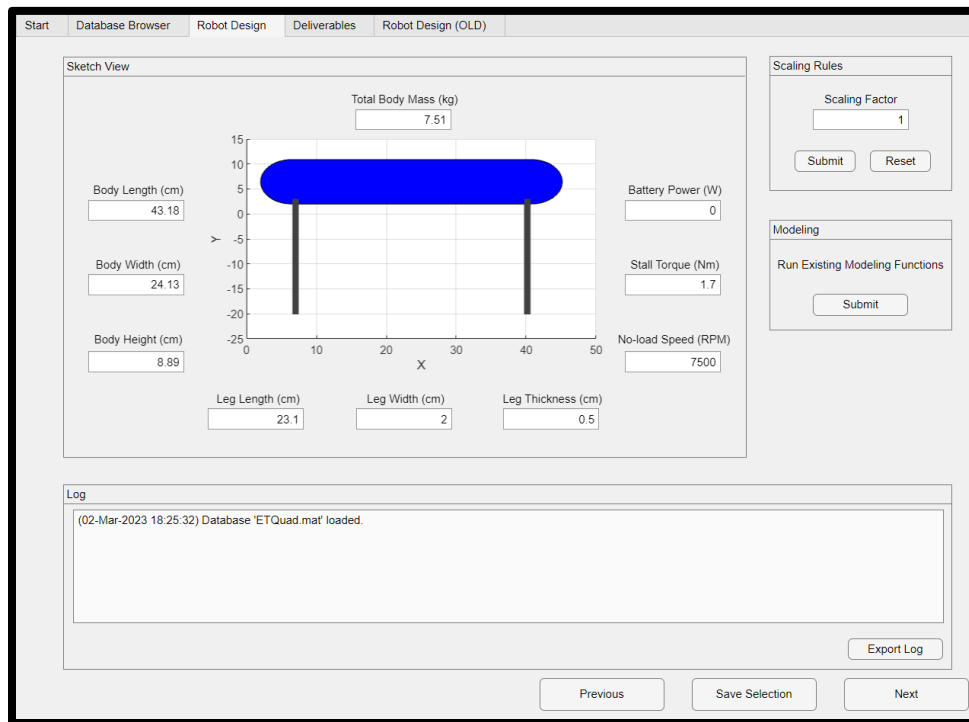
Click “Load Database” from either the File Menu or the lower portion of the Database Browser will allow you to select an appropriate “.mat” database file. When loaded, the Database will appear in the Available Databases pane. Select the database to show its detailed information as well as the included picture, as seen in Figure 2. Click the checkmark next to the database(s)

you wish to work with, then click “Save Selection” in the bottom right corner of the Database Browser to confirm and record your selections to your project file. Click Next to continue to the Robot Design tab.

Robot Design

The Robot Design tab acts as the primary center for working on and modifying your robot based on the database selected earlier.

*Figure 6
Robot Design Tab*

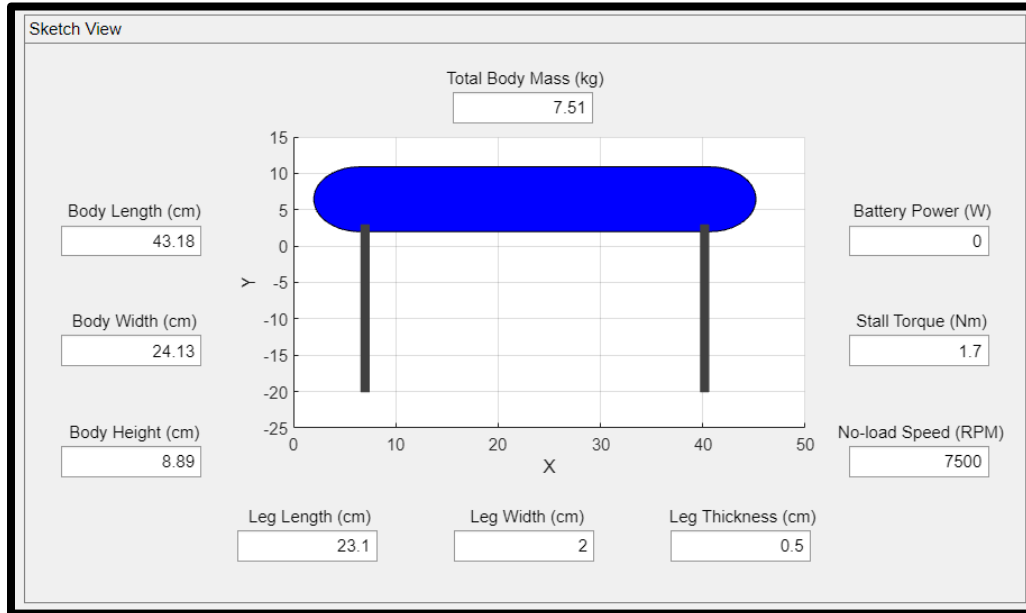


Note. The user can change different parameters of an old robot to see how the overall design changes.

Actions taken in each pane of this tab, as well as other actions throughout the tool, are recorded in the log at the bottom. To save this log for future reference, click “Export Log” at the bottom right corner. Note, this log will also be saved in your project file.

The Sketch View

Figure 7
Robot Sketch View

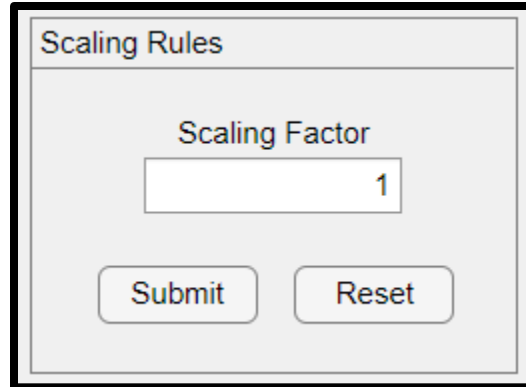


Note. The user can see a sketch view of the robot, which updates live as parameters are changed.

The Sketch View displays a simple 2-D sketch of the layout and proportions of the robot, with its physical characteristics listed underneath and to the left. When performing scaling operations, the 2-D sketch of the robot will automatically update to reflect the new dimensions. To the right are attributes of the robot's motor and battery, used for the modeling operations.

Scaling Properties.

Figure 8
Dynamic Scaling

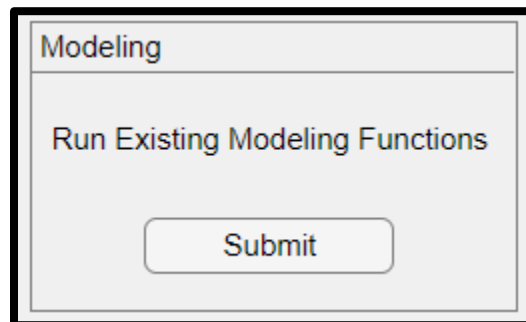


Note. The Dynamic Scaling laws used are based on length scaling.

To the right of the sketch view is the Scaling Rules pane. Entering in a scaling value here and pressing “Submit” will allow you to size the robot based on the laws of dynamic scaling. The physical dimensions of the robot will update both numerically in their associated fields, as well as in the sketch view. Once scaling has been performed, the Reset will become enabled, allowing you to set the scaling factor back to 1, and resetting the state of the robot to the initial database.

Running Calculations

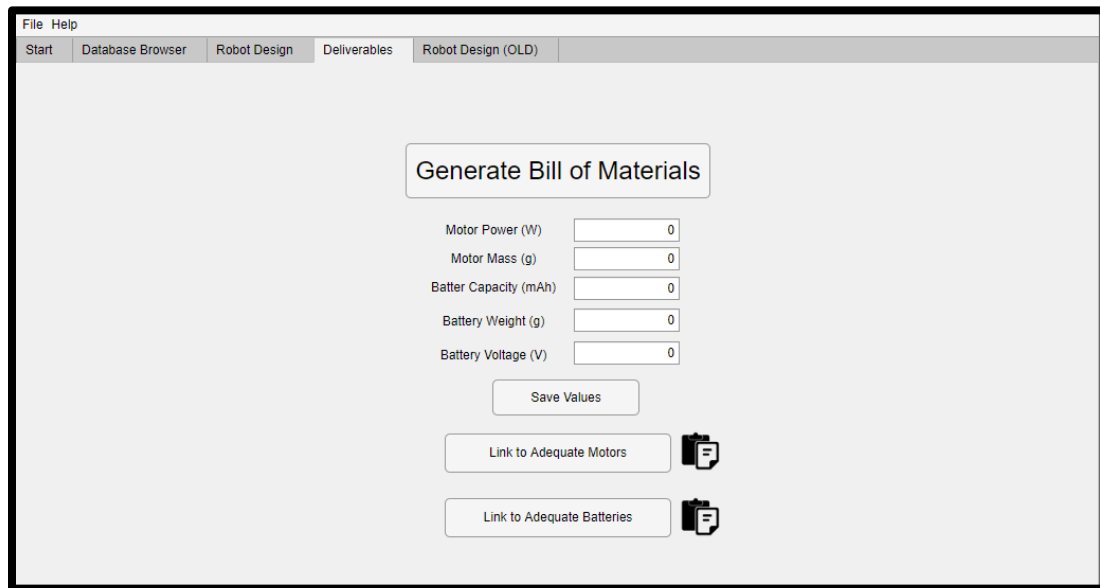
Figure 9
Run Simulink Analyses



To run the current suite of Simulink models through System Composer, select the “Submit” button in the Modeling pane. These calculations will use the mass listed in the Total Body Mass (kg) field at the top of the sketch view for the robot. A progress bar will appear once this option is selected, so please wait as running these models can take some time. Once complete, the motor and battery attributes in the sketch view will update accordingly.

Deliverables (Work in Progress)

Figure 10
Deliverables Tab

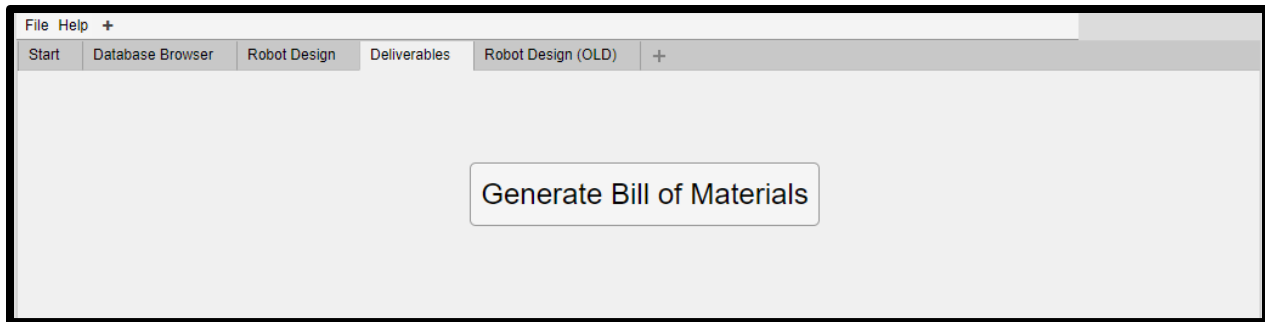


Note. The deliverables tab is currently a work in progress, but we hope to return critical parameters, as well as links, to help reduce the time to order.

After calculations using model functions have been performed and values for specific design parameters such as motor power or battery mass, you then have full functionality of the deliverables tab.

Bill of Materials

Figure 11
Generating a Bill of Materials



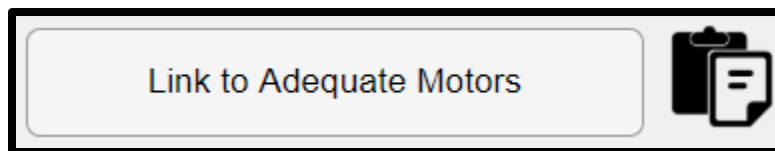
To populate the text boxes, select the “Generate Bill of Materials”. The text boxes will be populated with values to be later used in the process of generating links to a variety of sources to purchase materials from. To save these values for later, select the “Save Values” button in the middle of the screen. These values will be saved locally, allowing you to reference them later.

Figure 12
Saving Values



To begin looking at purchasing options for components generated from the bill of materials, select the “Link to” button for the specific component you are interested in. These buttons open your browser directly to the products page of the specific vendor CISCOR has used historically for those components. If you would rather save copy the link, the “Copy to Clipboard” functionality button is located directly to the right of the link.

Figure 13
Website links to motor



Web Resources

As stated above, all links generated are to the products page of vendors that CISCOR has used in the past to purchase these components. To further assist in your search for the proper

part, filters have been applied directly within the links to narrow down search results on their catalog pages. Currently, motors are being purchased from Maxon (www.maxongroup.us) and batteries are being purchased from MaxAmps (www.maxamps.com). As more board component types are added to the bill of materials functionality, they will be searched through McMaster-Carr (www.mcmaster.com).

Architecture and Models

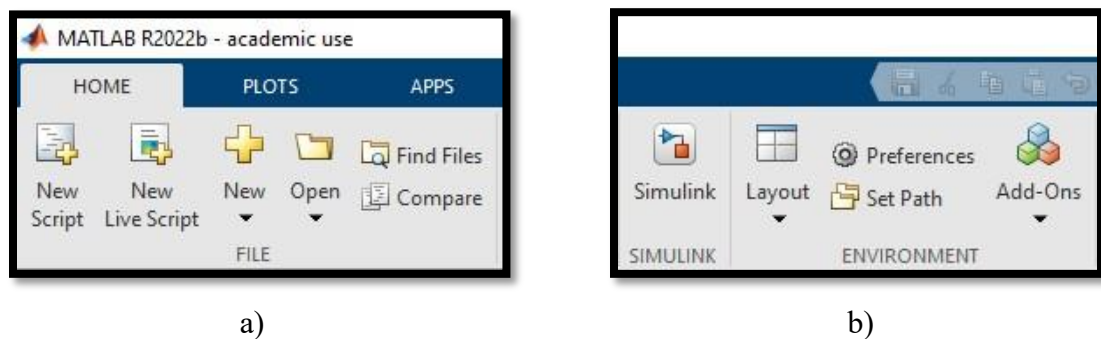
The following sections explain how the System Composer architecture is set up and how to change it, and they explain how to find and update the underlying Simulink models.

System Composer

Architecture Overview

There are several ways to launch System Composer. You can launch MATLAB and navigate to the Home Tab and the Simulink group, shown in Figure 14a and Figure 14b, respectively. Press the Simulink button.

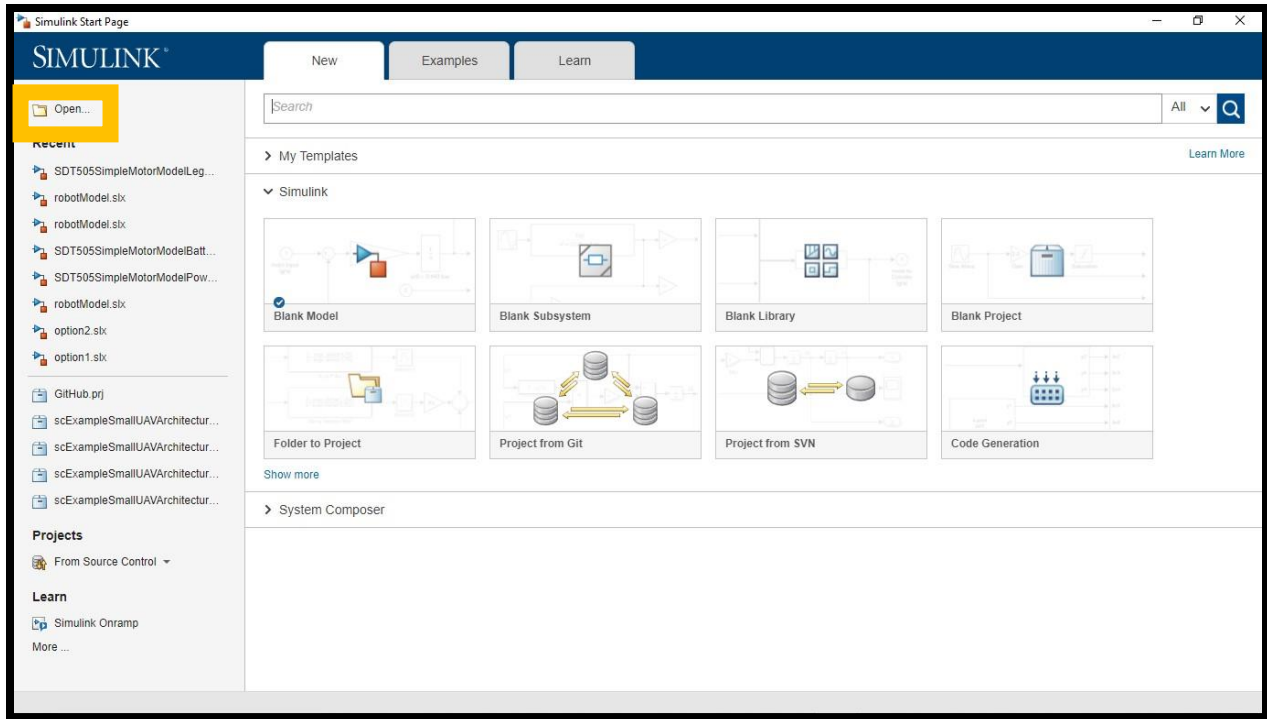
*Figure 14
Launching Simulink*



Note. a) shows the Home Tab in MATLAB where the Simulink group and button (b) are located. The Simulink button launches Simulink, and

Figure 15 shows the start page. It defaults to the new tab, but you can press open in the top left, shown in orange, to open an existing architecture. This project's architecture is called "robotModel.slx." To start a new architecture, you can open the System Composer dropdown, highlighted in red. For the robot software design tool, the user should only have to open the existing functional architecture not make a new one.

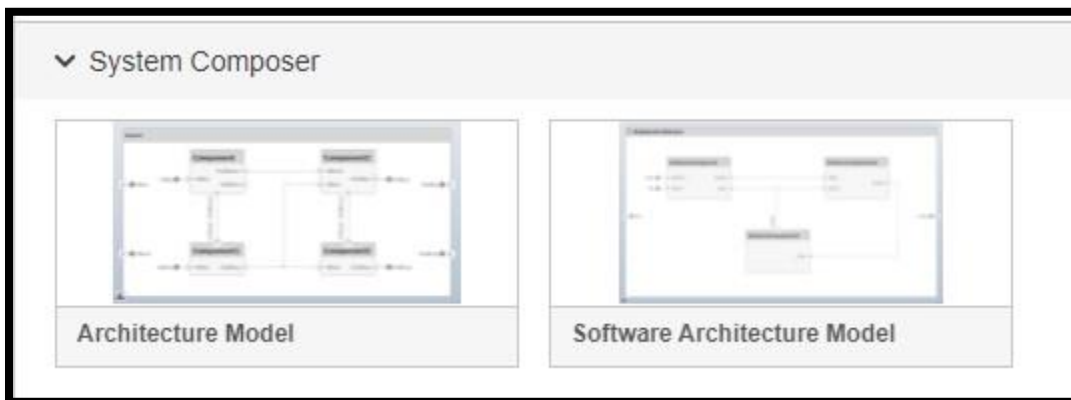
*Figure 15
Simulink Start Page*



Note. The Simulink start page shows recent files as well as different templates for creating a new model.

There should not be a need to create a new functional architecture, but Figure 16 shows the templates for doing so. The user can create an Architecture Model or a Software Architecture Model.

Figure 16
Start a new System Composer Architecture



Note. This figure shows the templates for creating a new System Composer Architecture.

The architecture file called “robotModel.slx” should now be open. The current functional architecture consists of three components: motor, battery, and leg links. The components are represented graphically as boxes within a larger box that represents the overall system. Figure 17 below shows the system with its smaller component boxes.

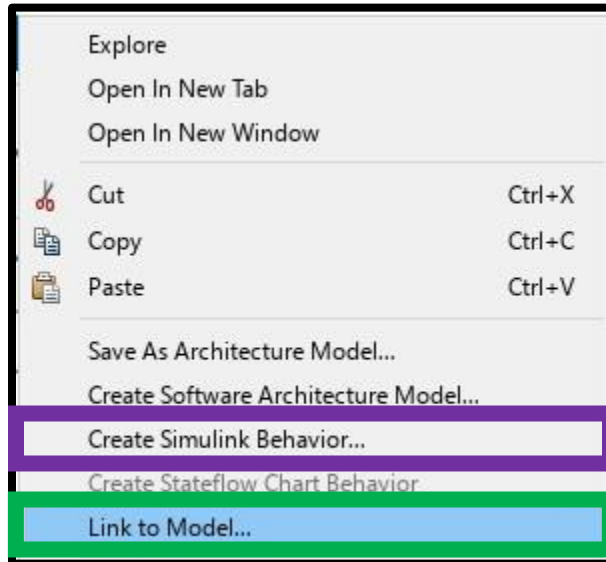
Figure 17
System Composer Architecture



Figure 17 – A picture of the System Composer components used in the design tool.

The blue and orange Simulink logo is in the top right of each component in Figure 17. This shows that there is a Simulink model attached to the System Composer component. To access the underlying model, double-click on the component. This opens the model in Simulink and allows you to modify it, which will be described in more detail in the next section. If an existing model is not attached to a component, right-click on the component and click “Create Simulink Behavior.” If the Simulink model already exists, select “Link to Model”. Figure 18 shows both these options in purple and green, respectively.

Figure 18
Linking to Simulink Models



Note. The user has the option of creating new models or linking existing ones to System Composer components.

Component blocks also have interfaces between each other. Figure 17 shows one such interface between the motor and battery components. This allows data from the components to be shared between them. These ports extend into the embedded Simulink models, but the Simulink side will be discussed in more detail later. The interfaces can be created before or after a Simulink model is attached to a component; however, it is important to note that attaching an existing Simulink model to a component removes any ports created before the model was attached. This is not an issue if a new model is created and attached.

Simulink

Modeling Assumptions

To simplify our analyses, we made several analyses based on recommendations from our sponsor Dr. Clark. We document these now so the user can better understand the context of the tool's outputs and so these assumptions can be later updated.

Our tool uses ET-Quad, a quadrupedal robot built by CISCOR, as the primary focus of our database and validation. ET-Quad uses a trot and 60% duty cycle for its walking gait at 2.5 Hz. A trot means that the diagonal pairs move together, and a 60% duty cycle means that the foot is on the ground, in stance, for 60% of the stride time. The stride frequency is 2.5 Hz, so a complete stance and flight cycle is completed at 2.5 Hz. This information is used for calculating the torque and speed requirements of the motors.

We also assume that the force felt by a foot is 3 times the weight of the robot divided by the number of legs on the ground:

Equation 1
Force on the foot

$$F = \frac{3mg}{\text{Number of Legs in stance}}$$

Since we assumed a trot gait, the force on a foot becomes 1.5 times the weight of the robot. We also assumed a leg length of 0.2 m to start our Simulink analyses, but this is likely to change as we update the model. The tool will use the values input from the user or from loaded databases. The torque calculations also need a touchdown angle to determine the stance torque. For the purposes of our analyses, we assumed this angle to be 30° measured clockwise from the horizontal. We also assumed that the foot's angle velocity followed a trapezoidal trajectory. The foot accelerates and decelerates each for half of the stance phase, and the angular velocity is constant during flight. Using our duty cycle, the acceleration and deceleration each take 0.12 s. The angular acceleration is linearly approximated using the change in angular velocity over the change in time from stance and flight. This angular acceleration is used in our calculation for stance torque.

We plot the speed and torques in flight and stance and use them to determine the motor's torque-speed curve. Real motors are not linear, but, for simplicity, we assumed that the torque

speed curve is linear. This allows us to determine the no-load speed and stall torque, which are used to describe a motor's specifications. On this linear speed curve, it is assumed that the middle of the curve represents the peak power output.

When concerning the batteries being used in our model, we assumed them to be LiPo batteries and only 80% of the total charge is being used. The power required by the motor is also assumed to be the power the battery needs to generate.

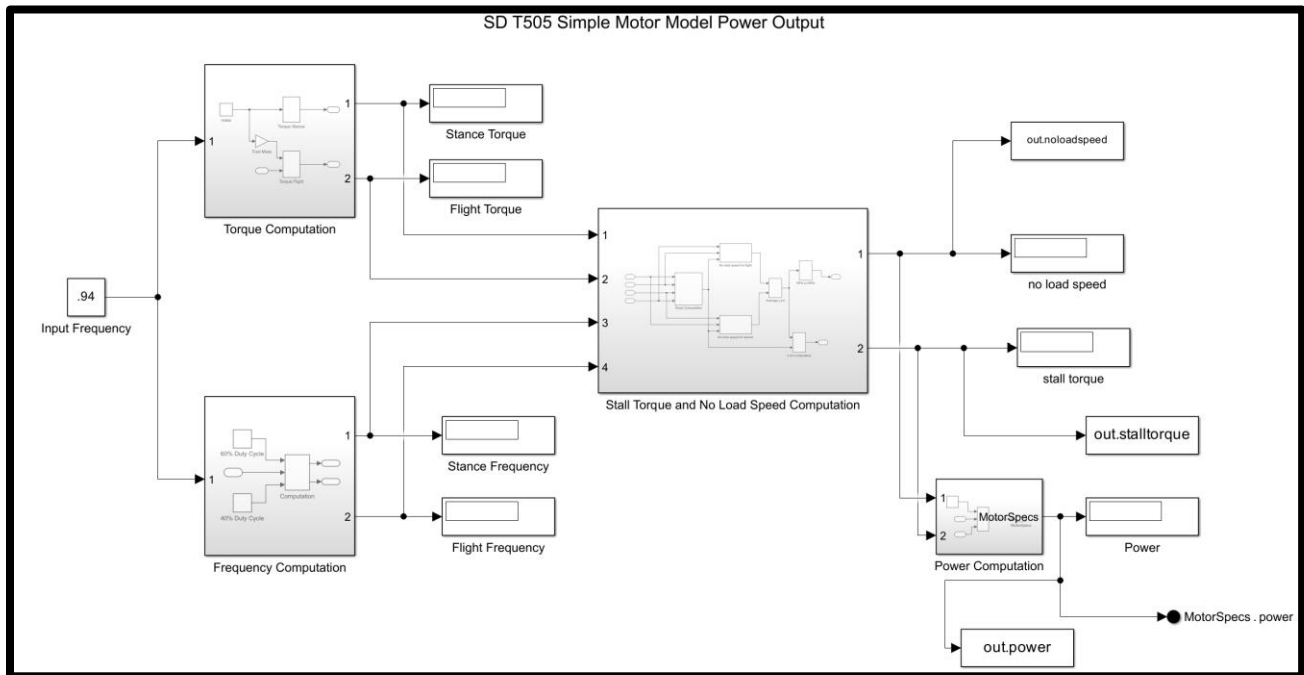
Motor Model

Figure 19 below displays the architecture of the motor model. The simple motor model is split into a flight phase and stance phase. For each of the phases the frequency and the torque are also being calculated. The computations for each phase can be found within each labeled subsystem and displayed through the labeled display blocks. When looking further into the computation blocks and smaller subsystems, they are labeled according to the calculations the blocks are performing.

The torque and frequency values for both phases are used to determine the stall torque and no-load speed. These values are also shown through display blocks. Each of these values are then used to find the motor's power output.

Figure 19
Motor Model Architecture

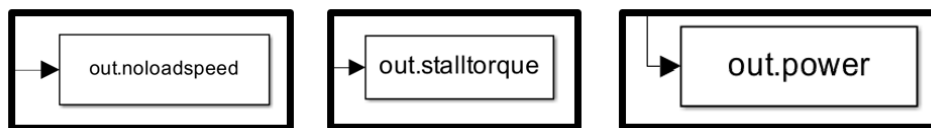
Figure 19



Note. A picture of the Simulink Motor Model.

The block components in the motor model shown below in Figure 20 are used to send values back to the MATLAB workspace. The no-load speed, stall torque, and power are all calculated values based on the user input that are being displayed on the GUI.

Figure 20
To Workspace Components



Note. A picture of the to workspace block elements used in the Motor Model.

In the bottom right corner of Figure 19, there is a MotorSpecs.power out bus element. This is used to export the power value calculated from the no-load speed and stall torque to other components found on the system composer interface. In this case, the motor model is exporting

the power value to the battery model. Below demonstrates the out bus element being used in the Simulink Motor Model.

Figure 21
Out Bus Element

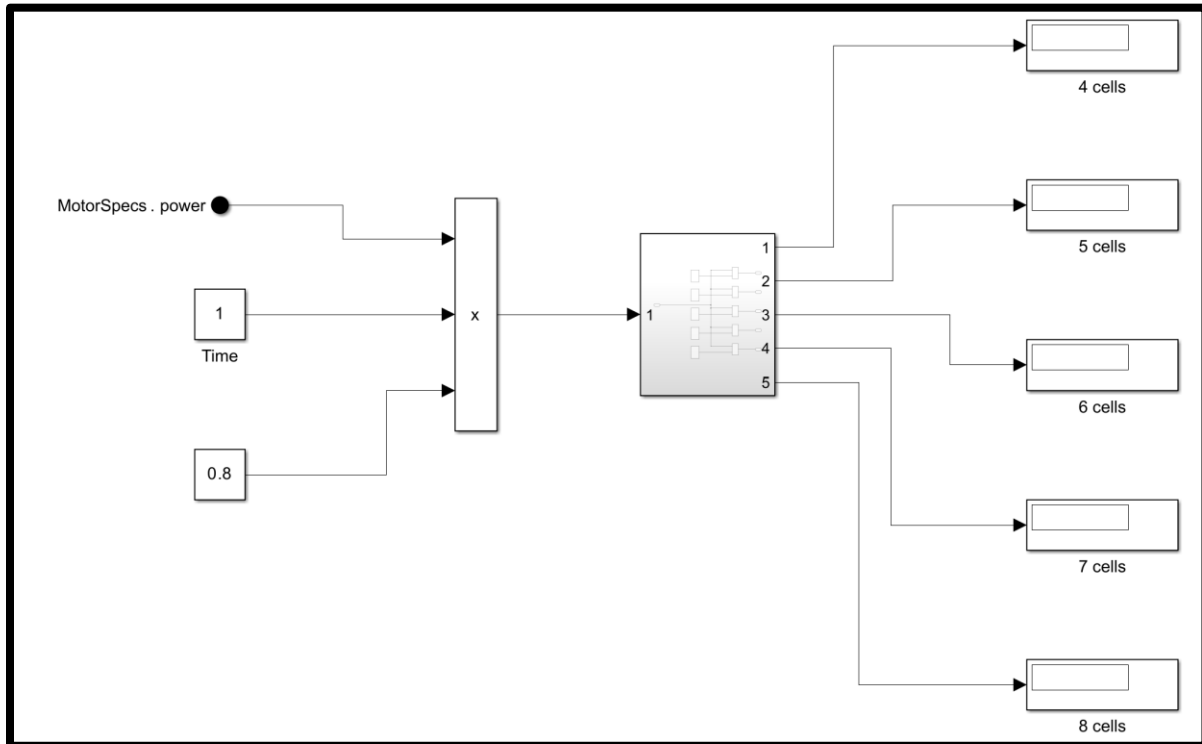


Note. A picture of the Out Bus Element used in the Motor Model.

Battery Model

Figure 22 shows the battery model. It takes in the power required by the motor from the `MotorSpecs.power`, which we assume is the power that the batteries need to supply. The 0.8 denotes an assumption that we will not drain the battery more than 20% and the Time block denotes out runtime requirement. These feed into a product block that multiply them together. From there, the model calculates the capacity in amp hours that will be required for batteries with different numbers of cells.

Figure 22
Battery Model Architecture

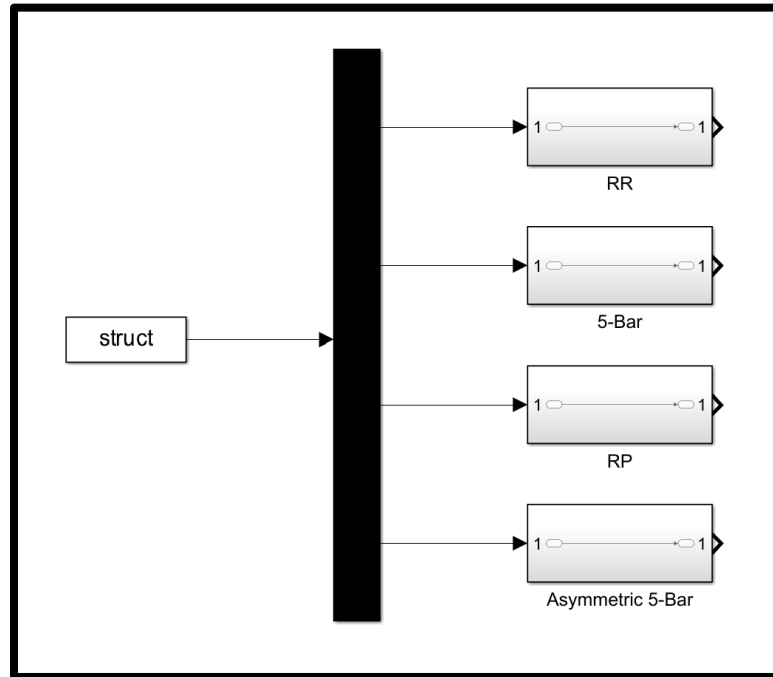


Note. A picture of the Simulink Battery Model.

Leg Linkage Model

The leg model shown below in Figure 23 uses a Demux block to split vector signals into scalar values. The struct going into the Demux block is taken from the MATLAB workspace that is determined from the user input. Each subsystem is labeled with the designated leg designs that the user can select when designing their robot. The complex equations pertaining to each leg design are not yet inputted into each subsystem. However, the framework for further development is set up to do so.

Figure 23
Leg Linkage Model Architecture



Note. A picture of the Simulink Leg Linkage Model.

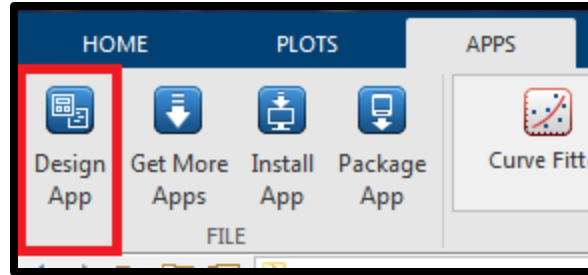
Source Code

MATLAB GUI Code

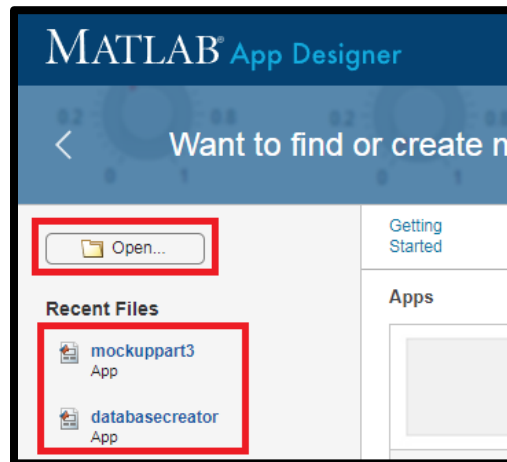
This tool is open source and intended to be adjusted and modified towards the needs of its users. Coding for the user interface and basic calculations in this tool are handled through the MATLAB App Designer interface, which allows for the graphical and programmatic modification of this tool.

MATLAB App Designer

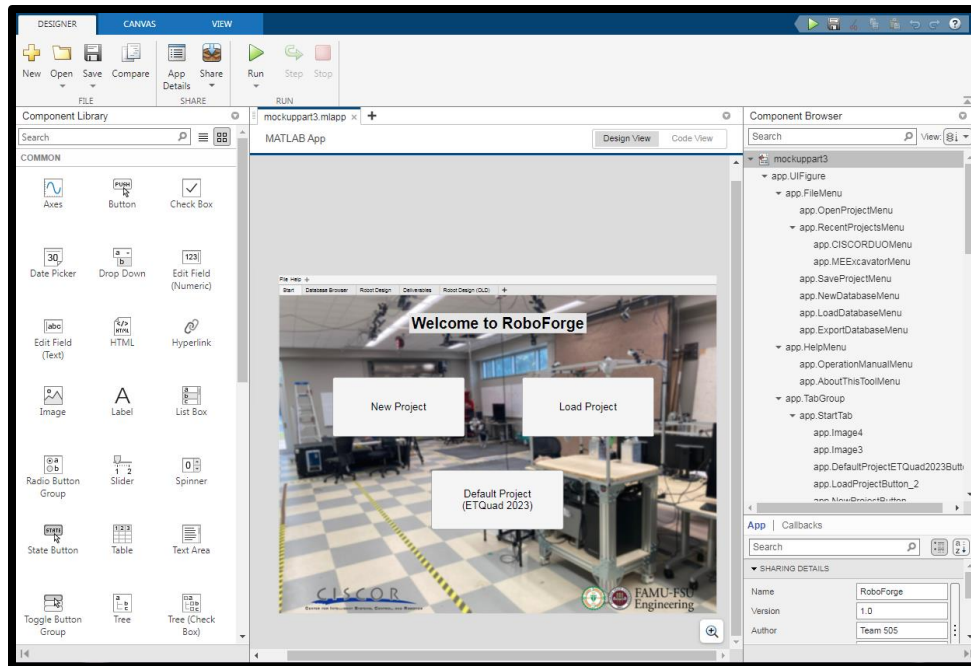
The App Designer can be launched from within MATLAB by switching to the Apps ribbon and selecting “Design App”. You can also launch the App Designer by selecting a “.mlapp” MATLAB App file from the Current Folder pane.



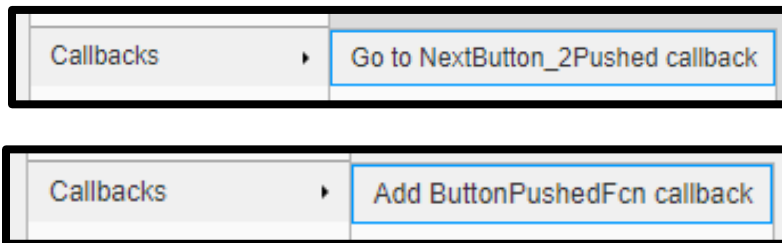
Once selected, the App Design loads into the Start Screen. To work on this tool, either select “Open” to choose the file, or select the tool from the “Recent Files” listed right below.



By default, once a file has been selected, App Designer opens to the Design View. This shows the current app in the center, with the Component Library to the left, and the Component Browser to the right.



To add a component to the app, drag the component from the library panel, and place it in the main app screen. To add a particular behavior to a UI (User Interface) element, right click the element and highlight the “Callbacks” option. This will give you the option to either go to the specific location of the callback in the code, or add a callback if one has not already been made.



Callbacks are functions tied to each element of the UI, and are typically activated upon clicking the element. Standard MATLAB code works in these callbacks, with some minor differences to be aware of, as compared to regular MATLAB coding:

- The App Designer is a fully object-oriented interface. Callbacks, other functions, and variables can be specified to be private or public. These elements are considered objects

of the main app, and so typical formatting for elements will appear as “app.element.value”.

- Apps have their own independent variable Workspace that is different from the “base” workspace used in MATLAB and Simulink by default. This app workspace is not visible, however. It is recommended to keep track of your variables and values through visual elements.

For details on the function of each particular app component, please refer to the MATLAB Help documentation. From now on, the functions referenced are specific to this tool.

Custom Functions

The following are a selection of functions central to the operation of this tool.

- Robotgrapher
 - Used to update the sketch view based on the currently selected database
 - Called when the current model is updated via scaling

```
% Handle all the graphing in the Robot Design tab
function robotgrapher(app,robotinfo)
% Clear whatever was already in the plot
cla(app.UIAxes);
% Plot the robot
% Robot body
body = rectangle(app.UIAxes,'Position',[2, 2, robotinfo.body.length,
robotinfo.body.height], 'FaceColor','b', 'Curvature',1);
% Robot legs
foreleg = line(app.UIAxes,[7,7], [3,3-
robotinfo.leg.length], 'linewidth',5, 'color','#404040');
rearleg = line(app.UIAxes,[robotinfo.body.length-3
```

```

robotinfo.body.length-3 ...
], [3 3-robotinfo.leg.length], 'linewidth', 5, 'color', '#404040');
% Update the side values
app.LegThicknesscmEditField.Value = robotinfo.leg.thickness;
app.LegWidthcmEditField.Value = robotinfo.leg.width;
app.LegLengthcmEditField.Value = robotinfo.leg.length;
app.BodyHeightcmEditField.Value = robotinfo.body.height;
app.BodyLengthcmEditField.Value = robotinfo.body.length;
app.BodyWidthcmEditField.Value = robotinfo.body.width;
app.TotalBodyMasskgEditField.Value = robotinfo.body.totalmass;
app.StallTorqueNmEditField.Value = robotinfo.motor.stalltorque;
app.NoLoadSpeedRPMEditField.Value = robotinfo.motor.no_loadspeed;
end

```

- **DynamicScale**
 - Function which governs and applies scaling rules
 - Current functionality passes the entire temporary database to this function, and returns the database with unedited portions copied over.

```

function [newModelParam] = dynamicScale(app,modelParam,a_length)
% This function dynamically scales the parameters in the struct
modelParam
% according to the length scaling factor a_length
newModelParam.body.length = modelParam.body.length * a_length;
newModelParam.body.height = modelParam.body.height * a_length;
newModelParam.body.width = modelParam.body.width * a_length;
newModelParam.leg.length = modelParam.leg.length * a_length;
newModelParam.leg.width = modelParam.leg.width * a_length;

```

```
newModelParam.leg.thickness = modelParam.leg.thickness * a_length;
newModelParam.body.totalmass = modelParam.body.totalmass * a_length^3;
% Return duplicates of the other fields
newModelParam.name = modelParam.name;
newModelParam.description = modelParam.description;
newModelParam.motor.mass = modelParam.motor.mass;
newModelParam.motor.stalltorque = modelParam.motor.stalltorque;
newModelParam.motor.noloadspeed = modelParam.motor.noloadspeed;
newModelParam.battery.capacity = modelParam.battery.capacity;
newModelParam.battery.mass = modelParam.battery.mass;
end
```

- LoadDatabaseMenuSelected
 - Handles importing a database and distributing it throughout the tool
 - Creates two copies of the imported database: editingdatabase as the temporary model to be worked on, and editingshadow which is used as a control to reset to
 - Automatically builds the database description based on its contents
 - Passes the database to Robotgrapher to update the sketch view

```
% Load the database from file
loadingdatabasefile = uigetfile(".mat"); % This gets the filename
% Bring the window back into focus
app.UIFigure.Visible = 'off';
app.UIFigure.Visible = 'on';
load(loadingdatabasefile); % load the database
app.editingrobot = robot; % Put the database into a temp struct that
we work on
app.editingshadow = robot; % Put the database into a second struct to
```

```
reset to later

% Grab some metadata for later
robotmeta = dir(loadingdatabasefile);

% Grab the names of all the database fields
editingdatabasenames = string(fieldnames(app.editingrobot));

% Build the description
robotdetailtext1 = sprintf("Database: %s \n\n",app.editingrobot.name);
robotdetailtext2 = sprintf("Available Nodes: %s, %s, %s, %s
\n\n",editingdatabasenames(1),editingdatabasenames(2),editingdatabasen
ames(6),editingdatabasenames(7));
robotdetailtext3 = sprintf("Description: %s
\n\n",string(app.editingrobot.description));
robotdetailtext4 = sprintf("Date Modified: %s \n\n",robotmeta.date);
robotdetail =
append(robotdetailtext1,robotdetailtext2,robotdetailtext3,robotdetailt
ext4);
app.TextArea.Value = robotdetail;

% Read data from the script and put it in a tree
firstdatabasenode = uitreenode(app.Tree);
firstdatabasenode.Text = app.editingrobot.name; % Database name should
be the 3rd entry

% Update the overview picture
app.Image.ImageSource = app.editingrobot.image;

% Call the robot plotting function
robotgrapher(app,app.editingrobot);
```

Database Structure

The following is the current layout of each entry of a robot database, as of the writing of this tool. Each robot database is stored as a structure withing a MATLAB “.mat” file. These databases can be created with the Database Creator tool launched from the Database Browser.

- Robot (Struct Main)
 - Name
 - Description
 - Image
 - Body
 - Total Body Mass
 - Sensor Package Mass
 - Length
 - Width
 - Height
 - Leg
 - Thickness
 - Width
 - Height
 - Mass
 - Motor
 - Mass
 - Stall Torque
 - Battery
 - Capacity
 - Mass

GitHub Access

A public GitHub repository contains the latest version of the code base. Click here for the [link](#). The repository is public to view but private to edit. To request editing access, please contact the CISCOR lab.