The Pennsylvania State University

The Graduate School

College of Engineering

A DESIGN METHOD FOR PRODUCT FAMILY TRADE STUDIES UTILIZING GVI AND PFPF METRICS WITH APPLICATION TO ROBOTIC GROUND VEHICLES

A Thesis in

Mechanical Engineering

by

Aaron Michael Bobuk

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2010

The thesis of Aaron Michael Bobuk was reviewed and approved* by the following:

Timothy W. Simpson
Professor of Mechanical & Industrial Engineering
Thesis Advisor

Sean Brennan
Associate Professor of Mechanical & Nuclear Engineering

Karl Reichard
Research Associate and Assistant Professor of Acoustics

Karen A. Thole
Professor of Mechanical Engineering
Department Head of Mechanical & Nuclear Engineering

*Signatures are on file in the Graduate School.

Abstract

Effective product platforms must strike an optimal balance between commonality and variety. Increasing commonality can reduce costs by improving economies of scale while increasing variety can improve market performance, or in our robot family example, satisfy a wide range of different missions. Two metrics that have been developed to help resolve this tradeoff are the Generational Variety Index (GVI) and the Product Family Penalty Function (PFPF). GVI measures the amount of product redesign that is required for subsequent product offerings to meet new requirements, whereas PFPF measures the dissimilarity or lack of commonality between design (input) parameters during product family optimization. GVI is examined because it is the most widely used metric applicable during conceptual development to determine platform components. PFPF is used to validate GVI because of its ease of implementation for parametric variety, as used in this example. This work describes a product family trade study that has been performed using GVI for a robot product family and compares the results to those obtained by optimizing the same family using PFPF. Additionally, this work provides a first attempt to validate the output of GVI by using a complementary set of results obtained from optimization. PFPF optimization is made possible by a fast, comprehensive, and accurate mathematical model that is developed as part of this work to calculate design parameters and functional capabilities of a robotic ground vehicle. Additionally, a design method for iteratively populating the trade space with robots using this model is presented. The results of this study indicate that while there are sometimes similarities between the results of GVI and optimization using PFPF, there is limited correlation between them. Moreover, the platform recommended by GVI is not necessarily the most performance-optimized platform, but it can help improve

commonality. In the same regard, PFPF may miss certain opportunities for commonality. The benefits of integrating the two approaches are also discussed.

Acknowledgements

I would like to first acknowledge my God and savior, Jesus Christ, who has given the confidence and assurance before Him to move forward into any difficult task, knowing that my relative success or failure does not determine my standing before Him. My worldly accomplishments can never compare to the wisdom, hope, fulfillment, and salvation found in Jesus Christ. The completion of this thesis is a result of that truth. I thank God both the opportunity to do this research and for the following people and sponsor:

I would like to thank my parents for their continual love and support. I am grateful for them providing me with the opportunity and giving me the motivation and enthusiasm to attend The Pennsylvania State University and carry out such work as this.

As my thesis advisor, Dr. Timothy Simpson gave me the opportunity to begin working on this project as an undergraduate student. I am grateful for not only this opportunity but for his equipping me with the knowledge to see this project to completion. I would like to thank Dr. Karl Reichard for supporting, overseeing, and guiding this project over the past few years. I am very appreciative of Dr. Sean Brennan, who first introduced me to Simulink® and for his extensive modeling guidance and support.

Numerous people within the project team and the EDOG lab have provided me with a significant amount of help and have taught me a great deal while working on this research. I would especially like to thank Laura Slingerland, Drew Logan, and Vipul Mehta.

This work was supported by the NAVSEA Contract Number N00024-D-02-D-6604, Delivery Order Number 0602. The content of the information does not necessarily reflect the position or policy of NAVSEA, and no official endorsement should be inferred.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview and Problem Statement

Product family design involves tradeoffs between commonality and performance – increasing commonality in the family provides multiple benefits (e.g., reduces development, procurement, inventory, and supporting costs), but too much commonality can yield suboptimal product performance. The benefits of commonality are many [1,2], and several examples of the pitfalls of commonality can be found in the literature [3]. For the robot product families considered in this thesis, shared parts and supporting systems and thus reduced inventory in the field is of great importance, yet commonality can translate into increased weight, decreased range or operating time, reduced mobility, or even excessive and/or unused functionality. Determining the appropriate level of commonality in a new product family, however, can be a difficult and challenging task [1,4].

A variety of methods and tools exist to help resolve the tradeoff between commonality and product performance in a product family. In this Chapter 6, two popular approaches for resolving this tradeoff are compared, namely, the Generational Variety Index (GVI) developed by Martin and Ishii [5] and an optimization-based approach that uses a commonality index – the Product Family Penalty Function developed by Messac, et al. [6]. As described in the next section, GVI provides a measure of the amount of product redesign that is required for subsequent product offerings, whereas PFPF measures the dissimilarity or lack of commonality between design (input) parameters during product family optimization. Both were applied independently to a family of robots (see Chapter 6), and comparison of the results provides the

first known attempt to validate the output of GVI by using a complementary set of optimization results.

1.2 Literature Review

1.2.1 Generational Variety Index

The Generational Variety Index (GVI) was developed by Martin and Ishii [5] to assist in identifying components within products that are likely to change over time in order to meet anticipated future market requirements. The value for GVI is based on an estimate of the required changes in a component due to uncontrollable factors including customer needs, reliability requirements, reduced prices, etc. [5]. GVI is computed using the seven step process outlined in Figure 1.

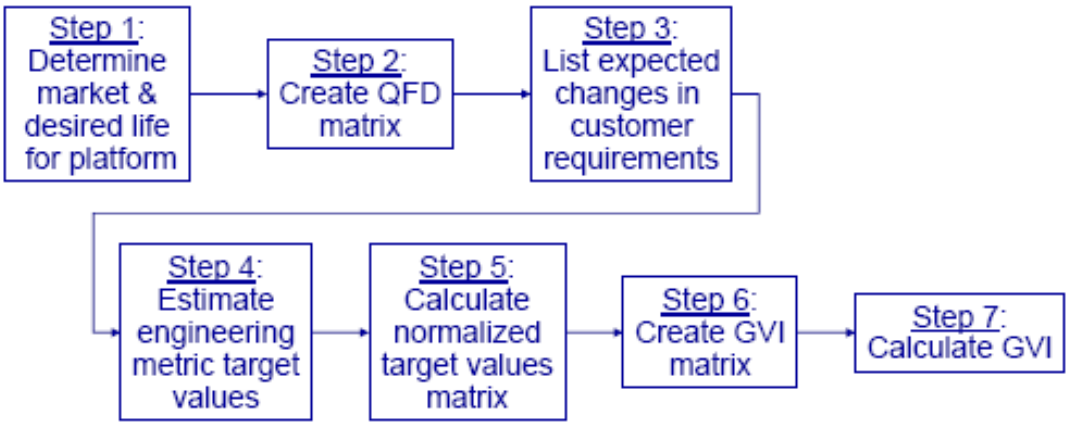

Figure 1: Steps to Compute GVI [5]

The first step in this process is to estimate the life of the product platform along with current and future market predictions. Then, two matrices are created. The first is the traditional Quality Function Deployment (QFD) matrix [7], which maps customer needs to engineering

metrics. The second matrix maps the engineering metrics from the first matrix to actual physical components used in the design of the product. Third, a column is added to the first QFD matrix to define the expected amount of change over the number of years specified in Step 1 qualitatively as high, medium, or low. This matrix visually describes how customer needs are expected to change. Next, target values for each engineering metric are added to the matrix for each timeframe that the product platform will be developed as described in the first step. The matrix is then normalized in order to graphically display the changes for the target values. The costs of changing components in order to meet engineering objectives on the product platform are evaluated as 0, 1, 3, 6, or 9 where a 0 indicates no changes are necessary and a 9 indicates that major redesign of the component would be necessary and cost more than 50% of the initial design cost. Table 1 provides descriptions for each of these values. The final step is to calculate the GVI for each component by summing each of the columns of the GVI matrix from Step 6. Components with low GVI scores require little redesign and can therefore be part of the platform, while high GVI scores indicate extensive redesign may be required; therefore, these components should not be part of the platform, but the interfaces to these components should be standardized to allow easy upgradability as the product evolves.

Table 1: Descriptions of GVI Component Scores [5]

Rating	Description
9	Requires major redesign of the component (>50% of initial redesign costs)
6	Requires partial redesign of component (<50%)
3	Requires numerous, simple changes (<30%)
1	Requires few, minor changes (<15%)
0	No changes requires

GVI has started to find applications in industry. Nomaguchi et al. [8] discuss the value in using perception-based approaches such as the GVI in concept design as a way to externalize design knowledge and further consider it. Unfortunately, despite its effectiveness, GVI is often criticized because the results are based on judgment and experience, which makes the metric subjective [9]. In addition to the subjectivity, the metric is poorly suited to capture change propagation across a modular platform because structural elements are much more strongly connected within a single product than across members of a family [10]. Moreover, modules are often designed to be functionally independent and thus have minimal interactions, limiting the propagation of change into other modules [11]. More recently, an Analytic Network Process (ANP) approach was created to measure the impact of design changes of modular products by measuring the relative change impact (RCI) among parts and modules [12]. Algorithms for clustering Design Structure Matrices (DSMs) have been used to identify modules [13]. Kang and Hong [14] note that GVI only offers binary information as to whether a characteristic should be differentiated or not; therefore, they developed a versatility measure to provide more information for the market. Alternatively, optimization can much more accurately represent the functional performance metrics and requirements of a product platform that depend more on functionality than a customer's ability to differentiate a product. GVI commonality is a binary yes or no decision on what to make common; however, as shown in Chapter 6, PFPF optimization can identify varying degrees of similarity.

1.2.2 Product Family Optimization

Product family optimization takes a completely different approach to solving the commonality-performance tradeoff by relying heavily on mathemetical models and optimization

algorithms to reach similar conclusions. Takai and Ishii [15] classify such evaluation methods into perception-based approaches (e.g., GVI) and analytic approaches (e.g., optimziation). Perception-based approaches such as GVI rely on experts quantifying their observations while analytic approaches calculate the performance of design concepts by optimizing performance functions [15]. As such, a mathematical model or other forms of performance functions relating design parameters must exist in order to use an analytic approach. The physics-based mathematical model for the robots considered in this thesis is presented in Chapters 2 and 3. In an analytical approach, commonality can be determined by modeling the product as an optimization problem where product performance across the family is considered, and tradeoffs are explored. Simpson [16] reviews more than 40 such optimization-based approaches that fall into this category of analytical approaches.

A challenge when using optimization is developing a suitable metric for commonality [17]. In this work, because we are considering parametric variations of the design (input) variables, we use the Product Family Penalty Function (PFPF), which measures the dissimilarity or lack of commonality in a family of products [6]. This metric allows commonality and performance to be appropriately balanced within the product family through the careful selection of common parameters that define the product platform. More specifically, PFPF helps determine which parameters should be held common (i.e., be part of the product platform) throughout the family and which should be varied during optimization.

PFPF works by penalizing design parameters that do not have a high degree of similarity throughout the product family while optimizing the desired objectives or fitness function for the family. PFPF is defined as the sum of the variability in the key design parameters normalized by the average value of each parameter [6]:

$$\text{PFPF} = \sum_{j=1}^{n} \frac{variability_j}{\bar{x}_j} \text{ where: } \bar{x}_j = \sum_{i=1}^{p} \frac{x_{ij}}{p} \text{ and } variability_j = \sqrt{\sum_{i=1}^{p} \frac{(x_{ij} - \bar{x}_j)^2}{(p-1)}} \quad (1)$$

where x_{ij} is the individual value of the i^{th} design parameter for the j^{th} product. Finally, n is the number of parameters being evaluated, and p is the number of products in the family that are being optimized. The variation is a percentage of the mean for the design (input) parameter in question so that while the parameters change during the optimization, the percent variation is always based on the mean of the variables [6].

1.3 Overview of Thesis

The mathematical model on which the robot product family optimization heavily depends is presented in Chapters 2 and 3. Chapter 2 provides a high level outline of the model by describing the tools used for modeling, integration of various software packages, simplification of the model by breaking it into subsystems, and how interdependencies are accounted for. Chapter 3 presents detailed descriptions of each subsystem within the model as well as each subsystem's corresponding equations.

Four existing robots are used to validate the model. The results of this validation are offered in Chapter 4. Upon generating over fifteen thousand robot designs, numerous design tradeoffs were discovered and presented in Chapter 5.

Chapter 6 provides and compares the results from the GVI and PFPF trade studies. A unique optimization method came out of this comparison. This unique optimization method combines the use of both GVI and PFPF. The method follows the steps outlined in Table 2. The

contents of this thesis present all of the relevant work necessary to carry out this optimization method for a family of robots.

Table 2: Optimization Method Combining GVI and PFPF Metrics

Step	Description
1	Perform GVI analysis
2	Create a mathematical model
3	Perform individual PFPF optimization to populate the trade space
4	Search the trade space for designs most closely resembling GVI commonality suggestions
5	Choose the best design: that which has the highest effectiveness and lowest PFPF

Chapter 2

Mathematical Modeling Overview

2.1 Modeling Overview

The physics-based mathematical model for this project was written in Simulink®, a simulation and model-based design package within Matlab® [18]. The Applied Research Laboratory (ARL) Trade Space Visualizer (ATSV) is used to visually steer and iteratively define inputs to populate new designs from the Simulink® model. In the subsequent text, the model inputs created by ATSV are referred to as user-defined inputs. ATSV is also used to view the robot designs populated by the model in order to examine tradeoffs. Both software packages and their applications are discussed in more detail in Section 2.2. A more detailed description of the interaction between software packages is discussed in Section 2.3. Additionally, the use of continuous versus discrete input values is discussed in Section 2.6.

The Simulink® model is partitioned into fifteen subsystems. Twelve subsystems are used to either calculate geometric parameters or size components for the robot, two are used to determine capabilities, and the final subsystem is used to determine how well the design meets a set of predefined threshold and objective requirements. Breaking the model into separate subsystems helps to organize the model diagram within Simulink® by grouping similar calculations together. Using a subsystem modeling approach also allows subsystems to be validated individually as shown in Chapter 4. This subsystem breakdown is further discussed in Section 2.4. The interdependent parameters that exist within the model are discussed in Section 2.5.

Equations coded into each subsystem are derived by either experimental testing and creation of best fit curves or by static kinematic analysis. While this model is capable of

providing accurate design representations as shown in Chapter 4, the intent was never to create a high fidelity model capable of predicting complex capabilities such as a three-dimensional reach envelope of a robot manipulator. Rather the intent of the modeling efforts were to be able to quickly populate a trade space consisting of thousands of design possibilities over a wide range of input parameters and evaluate basic capabilities. Figure 2 is a graphical representation of the level of design detail that this model provides. Rather than focusing on track patterns or suspension system design, the model outputs higher level parameters such as track width, battery dimensions, maximum vehicle velocity, etc. The ensuing chapter discusses in more detail how the equations in each particular subsystem are modeled and what they are capable of providing.

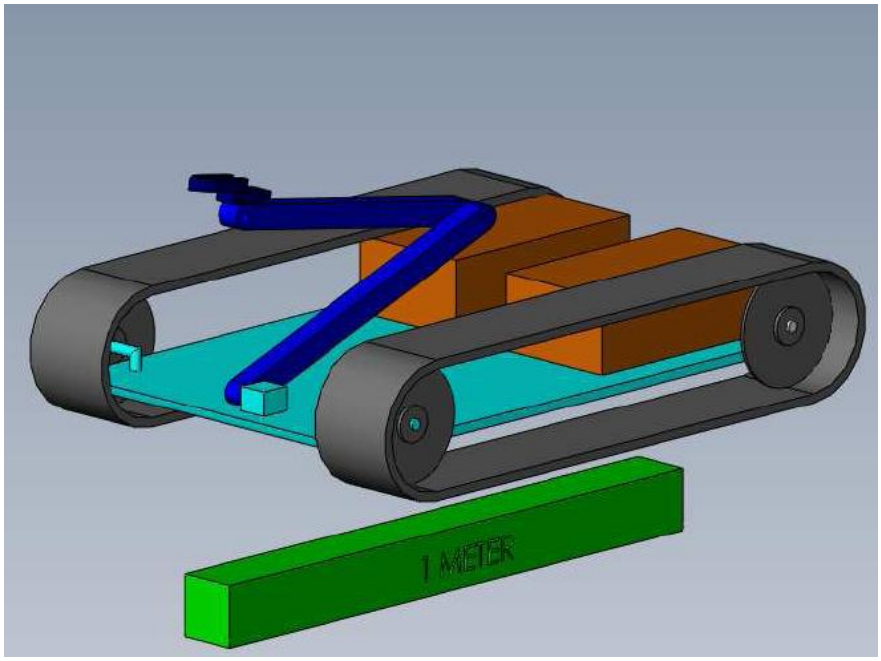

Figure 2: Graphical Representation of Robot based on Model Output

2.2 Modeling Tools

Simulink® is a mulitdomain simulation and Model-Based Design environment within Matlab® for dynamic and embedded systems [18]. Simulink® is used for several reasons. The primary rationale for use is Matlab®'s nearly universal understanding and use amongst the engineering community. An earlier version of this model was written in Mathematica®. The Mathematica® model presented challenges to the users because it was very difficult to debug and integrate with ATSV. This problem was alleviated in the switch to Simulink® due to its inherent block diagram format where an arrow is drawn from the output of an embedded function to the location it is used next. In additional to being able to trace variables as they are used throughout the diagram, it is also very easy for a user to attach a "scope" to any signal path and monitor how that particular variable converges with each iteration. Furthermore, breakpoints can be set within an embedded function, and values of all of the variables used within that function can be examined for error. Simulink® is also very fast. Whereas the previous generation Mathematica® model took well over two minutes to converge upon a single design iteration, the Simulink® model is capable of running a complete design iteration in approximately four seconds. It should also be mentioned that the bottleneck in the speed of the Mathematica® model was the link between Mathematica® to the solver built into Excel. Matlab® is able to generate an executable file that directly interfaces with ATSV, thus significantly speeding up the optimization process.

The Applied Research Laboratory's Trade Space Visualizer (ATSV) is used to sample the model's inputs and perform trade studies [19,20]. ATSV is a Java-based application created by Penn State that is connected to the mathematical model. Once the model is linked to ATSV, designs are generated "on-the-fly," giving the user control over the trade space exploration

process. ATSV allows users to explore the design space using random sampling, manual sampling and/or Pareto sampling, which implements Pareto-based Differential Evolution along the entire Pareto front [21]. The "in the loop" user can also visually steer the model with the use of an Attractor Sampler to search near a specific point, a Preference Sampler to search within an area of interest, and/or a Guided Pareto Sampler to search the trade space using a combination of attractor and preferences [22]. By combining multi-dimensional data visualization techniques with visual steering commands, the user is able to guide the optimization process while "shopping" for Pareto optimal design**s** [23]. ATSV is also used in the post processing of the data generated by the model. The software is capable of plotting data in multiple dimensions using such plots as glyphs, histograms, scatter matrices, and parallel coordinates [24]. Tradeoffs in design parameters are determined using these plots.

2.3 Software Integration

As mentioned in Section 2.2, the model for this work was created using Simulink®. An executable Windows® file was created from the Simulink® model using the Real-Time Toolbox in Simulink® (see Appendix A for the method used to create this executable file). This executable file was then embedded within another executable file that reads in a text file containing the model inputs, which are determined by ATSV, and outputs a text file containing the model outputs that are read into ATSV.

The basic interaction of the two software packages used to run the model is outlined in Figure 3. ATSV first generates values for the input variables between a user-defined upper and lower bound. These values are then written to a text file called "inputs.txt". ATSV then runs the packaged Windows® executable package which is boxed together in Figure 3. Contained within

this package is a Windows® executable file to provide an interface between ATSV and the executable Simulink® model. This executable interface within the executable package first writes the inputs from "inputs.txt" to a Matlab® .mat file called "inputs.mat". This step is necessary because the Windows® executable file created in Simulink® is incapable of reading and writing .txt files to the system. However, Simulink® inherently reads and writes .mat files when run. Upon loading the inputs to the .txt file, the Windows® executable Simulink® model is run. The model writes the outputs to another .mat file called "outputs.mat". The primary Windows® executable file then writes the .mat file to a .txt file called "outputs.txt". The output parameters contained within "outputs.txt" are then read and stored by ATSV. This completes one design iteration, i.e., one robot design is created ad analyzed. ATSV then generates a new set of input variables, and the process is repeated.

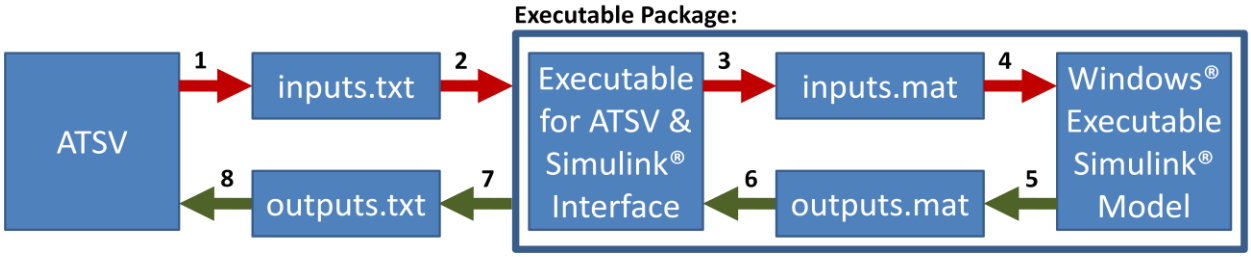

Figure 3: Software Interaction

2.4 Subsystem Overview

As mentioned previously, the robot model contains fifteen subsystems. A top-level block diagram of the entire model is shown in Figure 4. The subsystems are organized such that the subsystems requiring exclusively user-defined model inputs are oriented to the left most side of the model. These subsystems are evaluated first. The subsystems to the right depend on user-

defined inputs as well as outputs from other subsystems. The block diagram shown in Figure 4, outlines which subsystems are dependent on one another by mapping the subsystem output parameters to each subsystem where those parameters are used as inputs. The two parameters which are interdependent between subsystems (e.g., maximum velocity and total vehicle mass) can be identified in this figure by their left directed arrows into the two transfer functions which then point back into the subsystems which depend on them.

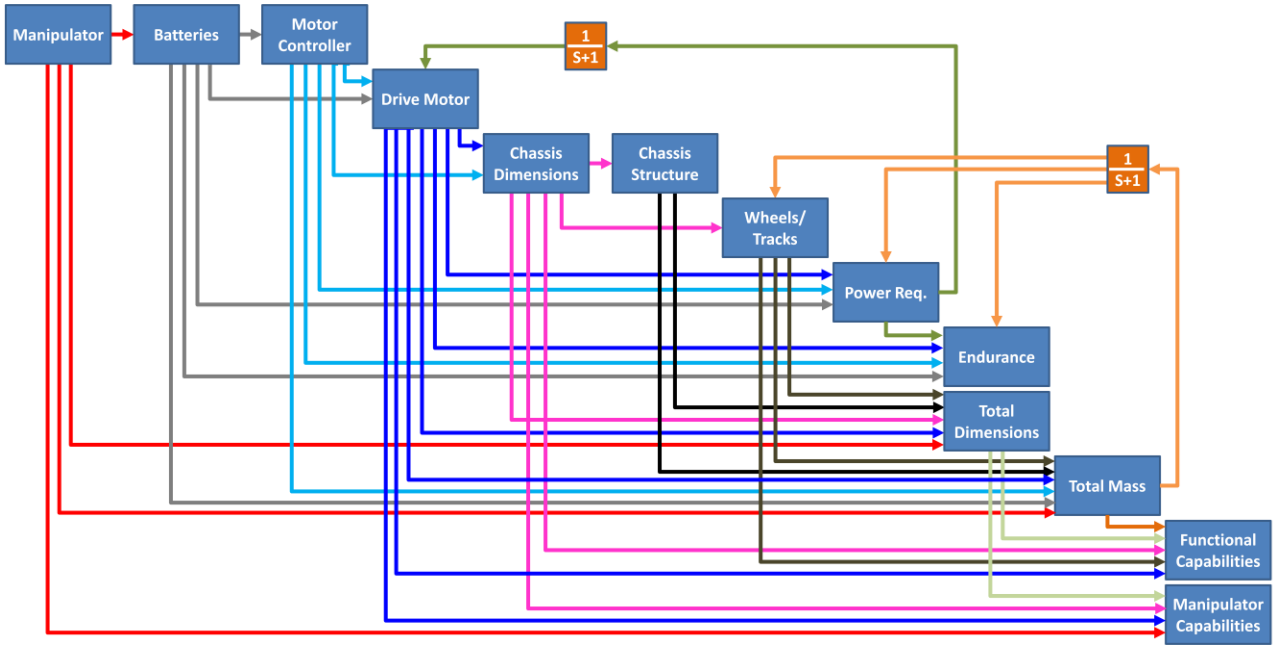

Figure 4: Top Level View of Simulink® Model

These fifteen subsystems, from the first to be evaluated to the last are Manipulator (see Section 3.1), Batteries (see Section 3.2), Motor Controller (see Section 3.3), Drive Motor (see Section 3.4), Chassis (see Section 3.5), Chassis Structure (see Section 3.6), Wheels (see Section 3.7), Tracks (see Section 3.8), Power Requirements (see Section 3.9), Endurance (see Section 3.10), Total Vehicle Dimensions (see Section 3.11), Total Vehicle Mass (see Section 3.12),

Functional Capabilities (see Section 3.13), Manipulator Capabilities (see Section 3.14), and Effectiveness (see Section 3.15). Contained within each subsystem is another sub-block-diagram containing embedded Matlab® functions and Simulink® operators as described in Chapter 3.

As previously mentioned, two variables with a right to left flow can be identified in Figure 4. These two variables are (1) maximum vehicle velocity and (2) vehicle mass. Both of these variables travel in the reverse direction in relation to the rest of the model because they must be solved for iteratively. The vehicle velocity must also be iteratively solved for within the Endurance subsystem. The chassis mass must be iteratively solved for within the Chassis Structure subsystem. These interdependencies are further discussed in the next section.

2.5 Interdependencies (Iterative Loops)

As mentioned in Section 2.4, the robot model contains two interdependent variables between subsystems. The first interdependent variable is maximum velocity. This interdependency first appears when sizing the drive motor gearbox. Additionally, vehicle velocity and chassis mass are interdependent variables within two separate subsystems.

As outlined in Figure 5, the Drive Motor subsystem depends on several user-defined inputs including the number of driven wheels, wheel diameter, drive motor configuration, bus voltage, and auxiliary power draw. The subsystem also depends on the total power available from the Batteries subsystem, and the motor controller efficiency from the Motor Controller subsystem. Finally, the Drive Motor subsystem depends on maximum vehicle velocity, which is not evaluated until much later in the chain of subsystems. This interdependency appears because a velocity must first be known to size the drive motor gearbox such that the motor operates at

peak efficiency. However, the efficiency losses from both the drive motor and gearbox are needed to accurately calculate the maximum vehicle velocity.

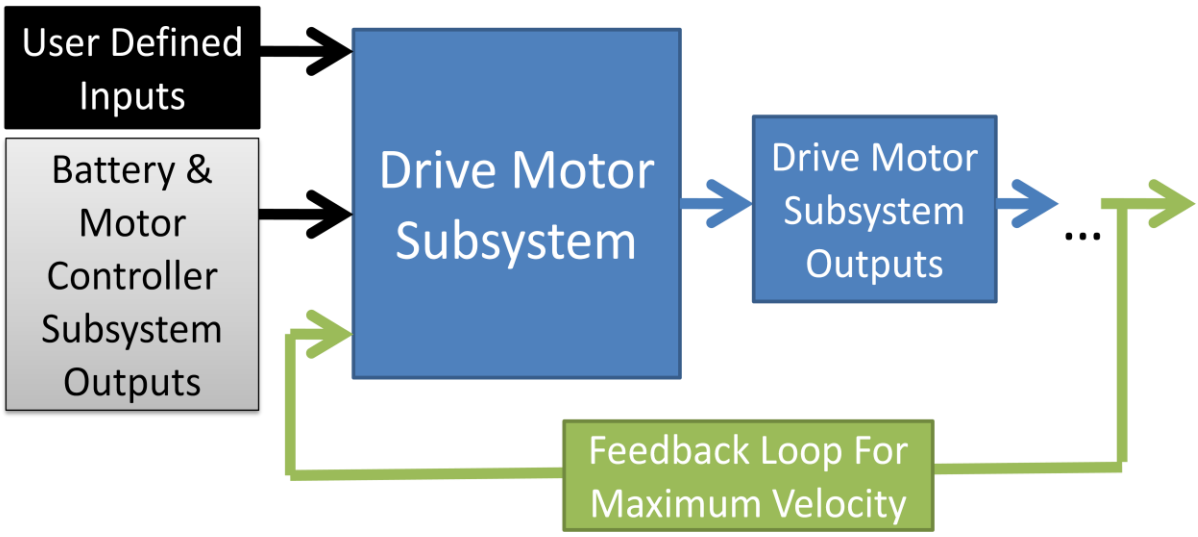

Figure 5: Drive Motor Subsystem Interdependency

The problem of circular dependency is accounted for with the implementation of a feedback loop. The feedback loop connects the maximum vehicle velocity output from the Power Requirements subsystem to the input of the transfer function shown in Figure 6. The output of the transfer function is then the input for the Drive Motor subsystem.

The integrator block in the state space transfer function shown in Figure 6 contains an initial guess for maximum vehicle velocity. The model uses this initial guess to size the drive motor as well as the subsystems between the Drive Motor subsystem and the Power Requirements subsystem. The model then calculates the maximum vehicle velocity and passes this value for maximum vehicle velocity through the dependent subsystems. This loop is repeated for a total of 40 seconds at 0.05 second time steps, thus resulting in 800 iterations. The

simulation time and time step are both specified in the configuration parameter window in Simulink®.

In order to be sure that all of the initial conditions can be user-defined, an all-integrator representation of a transfer function is used. A portion of the feedback loop for maximum vehicle velocity is shown in Figure 6. In this figure, the blocks highlighted in orange are collectively the all-integrator representation of the transfer function shown in Equation 2.

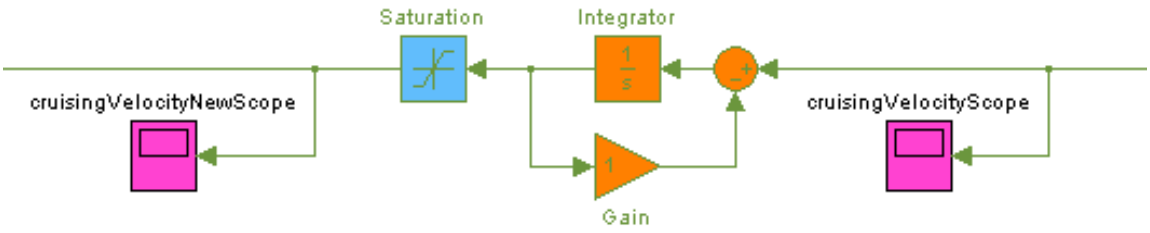

Figure 6: Vehicle Velocity Feedback Loop in Simulink

$$\frac{cruisingVelocity}{cruisingVelocityNew} = \frac{1}{s+1} \tag{2}$$

The feedback loop also contains a saturation block, highlighted in blue in Figure 6, to apply an upper and lower bound on the maximum vehicle velocity. This bound is set at 1000 m/s, and 0.01 m/s respectively. The only way that the feedback loop could hit one of these bounds is if maximum vehicle velocity were increasing without bounds and would never converge. If this would happen then the model either outputs nonsensical results or results in an error. The latter would cause the model to stop running during trade space exploration. If a design happens to hit either this minimum or maximum bound, then it is removed during post processing. Hitting this minimum or maximum bound did not occur during trade space

exploration; however, the saturation block should remain in the model to prevent possible error when sampling outside of the input bounds used for this particular trade space exploration.

The feedback loop shown in Figure 6 also contains two scopes. These scopes allow the user to monitor the convergence of maximum vehicle velocity. Using these scopes, model convergence can be proven using the final value theorem. To do so, the initial values for the three interdependent variables are guessed as shown in Table 3. The model is then run with a fixed set of inputs. Once the simulation is complete, the scope to the left of the transfer function labeled cruisingVelocityNewScope is examined. It can be seen from Figure 7 that maximum vehicle velocity initially begins at 2 m/s and then overshoots the final value of 7.52604317 m/s to which it eventually converges.

Table 3: Initial Values for Feedback Loops

chassisMassFeedback	10	kg
maxCruiseVelocityFeedback	2	m/s
vehicleMassFeedback	30	kg

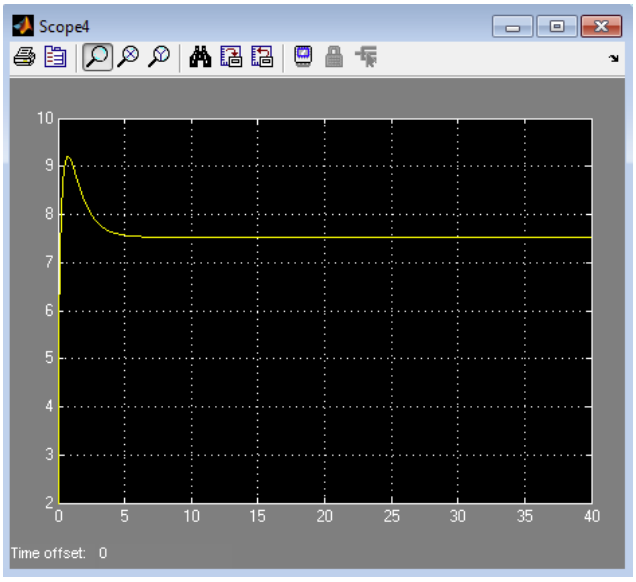

Figure 7: Maximum Vehicle Velocity Scope Using Initial Guesses

Figure 7 shows that the model is asymptotically stable [25]. Because the system is stable, the final value theorem is used to ensure local convergence of the model given a finite domain of input parameters [25]. This is done by using the same set of model inputs, but specifying the initial guesses for the three interdependent variables as the final values from the previous simulation. These values are shown in Table 4. Examining either scope after simulation, as shown in Figure 8, the maximum vehicle velocity begins and ends at precisely 7.52604317 m/s. There is absolutely no deviation from this value at any point during iteration, thus ensuring local convergence of the model given a finite domain of input parameters. Additionally, this process was used to ensure that the other interdependent variables are asymptotically stable and locally convergent.

Table 4: Calculated Values for Feedback Loops

chassisMassFeedback	13.53389737	kg
maxCruiseVelocityFeedback	7.52604317	m/s
vehicleMassFeedback	101.76047305	kg

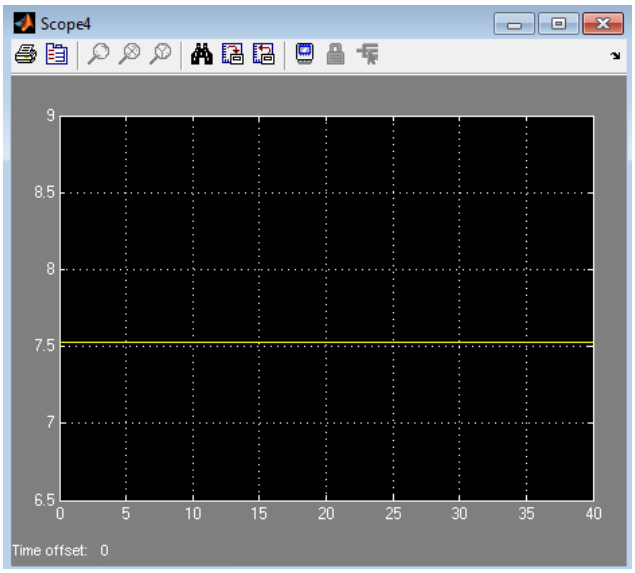

Figure 8: Maximum Vehicle Velocity Scope Using Calculated Values

In the next chapter, the Drive Motor subsystem, as well as the fourteen other robot model subsystems are explained in detail. The subsystems are presented in the order that they are evaluated within the model as described in Section 2.4.

2.6 Use of Discrete or Continuous Input Parameters

ATSV is capable of inputting either discrete or continuous input parameters to the model. Utilization of continuous values allows, for example, wheel diameter to be specified as 0.3724 [m]. Use of continuous parameters allows the robot to be optimized as much as possible; however, it is unlikely that an off-the-shelf wheel of precisely that diameter exists. Rather, a

wheel of this diameter would have to be custom made or a close substitution would have to be found. Custom parameters are not always possible due to development and production cost and time constraints.

On the other hand, discrete values can be used to allow only off-the-shelf components to be specified within the design. Discrete values for parameters such as wheel diameter can be specified directly in ATSV. On the other hand, utilization other discrete parameters involves modeling certain subsystems differently. For or example, the use of a standard BB2590U battery versus a custom Lithium-Ion battery is discussed in Section 3.2. The performance hit with the use of a BB2590U battery is shown in Section 5.3.

Chapter 3

Subsystem Modeling

This chapter provides a detailed description of each subsystem within the model. Each section within the chapter outlines the inputs and outputs of the subsystem at hand. Each section also explains the equations used in each subsystem and how each equation is related to another.

3.1 Manipulator Modeling

3.1.1 Subsystem Overview

The manipulator subsystem is the first subsystem to be evaluated in the model because all of the input parameters are user-defined. A high level view of the subsystem is shown in Figure 9. The inputs to this subsystem are the light blue boxes to the left of the subsystem 'black box', which is actually shaded green. The light blue coloring of the input boxes indicates that the parameters are user-defined inputs.

A diagram of the manipulator that is modeled in this subsystem is shown in Figure 10. This diagram also defines the nomenclature used throughout this chapter. As shown in Figure 9, inputs to the manipulator subsystem include primarily geometric parameters including the length of units [m] and mass of units [kg] defining the end-effector or gripper, lengths of each segment (L1A1Length, L2A1Length, L3A1Length) of units [m], density of the manipulator material [kg/m^3], and inner arm radii of each segment tube (innerArmRadiusL1A1, innerArmRadiusL2A1, innerArmRadiusL3A1) [m]. The subsystem also includes inputs that factor into the dexterity and mobility of the manipulator. These inputs include the dimensionless number of degrees of freedom at each joint (nDOFL1A1, nDOFL2A1, nDOFL3A1). The maximum weight that the manipulator is sized to lift, before tipping of the robot during a fully

extended reach is considered, is an input defined as FLift and has units of [kg]. Gearbox ratios for each motor (gearboxRatioWrist, gearboxRatioElbow, gearboxRatioShoulder, gearboxRatioTorso) are inputs to the subsystem. Finally, the number of links, ranging from 1 to 3, is an input.

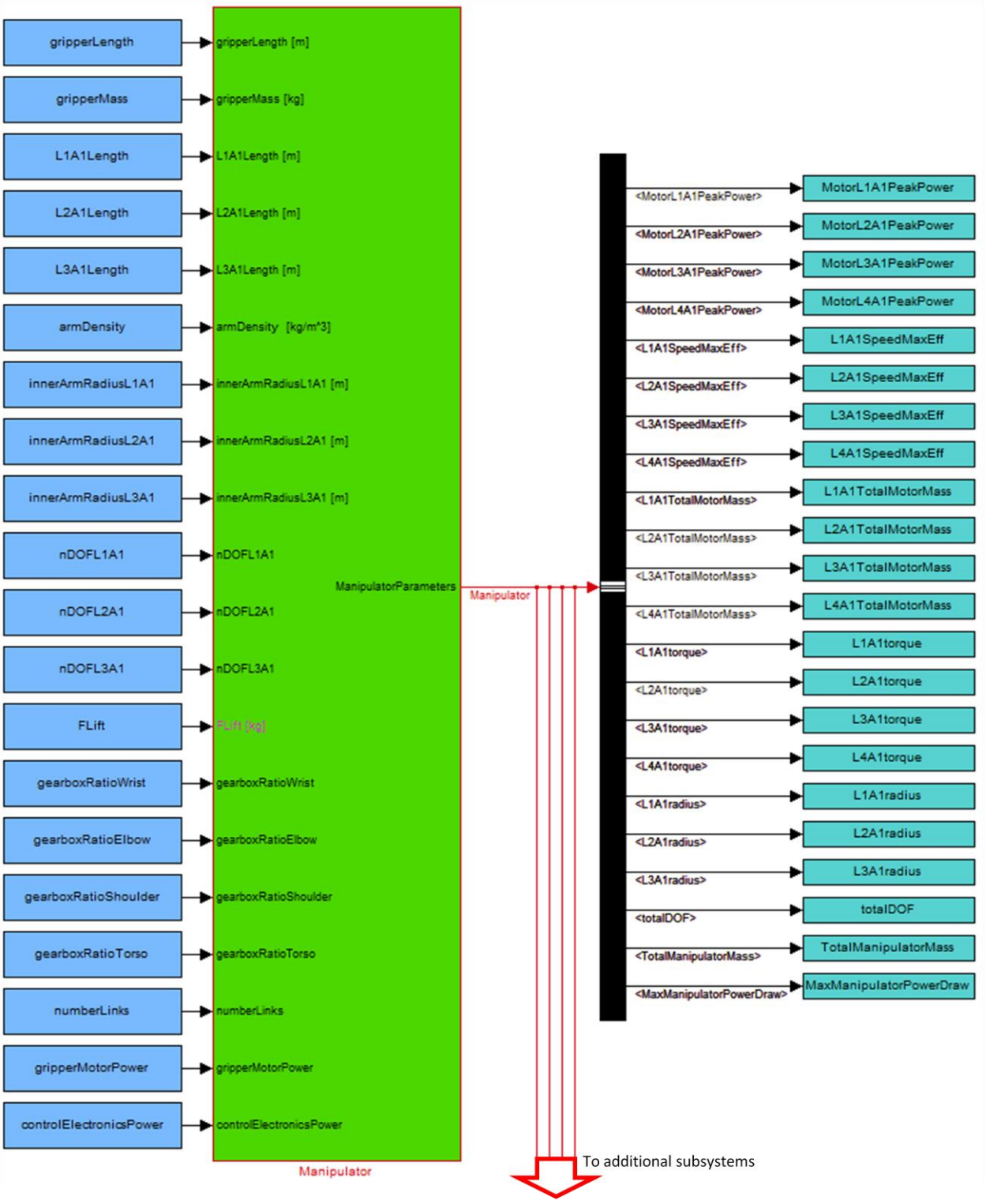

Figure 9: Simulink® Manipulator Subsystem

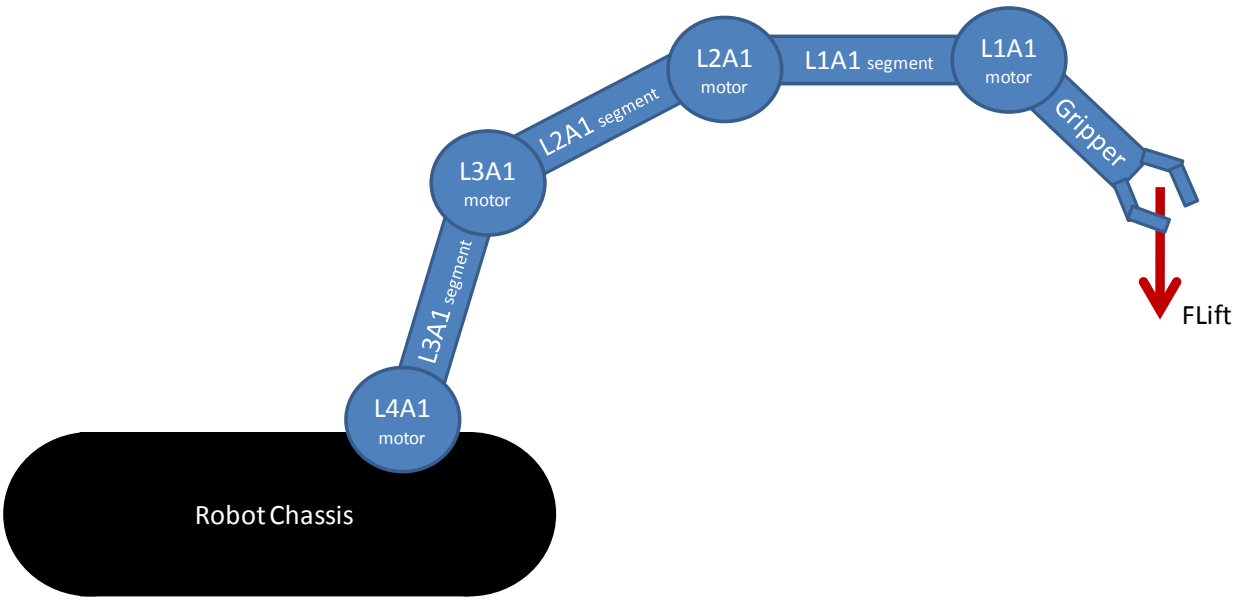

Figure 10: Manipulator Diagram Highlighting Nomenclature

As seen in Figure 9, the first group of outputs from this subsystem is the peak power [W] draw from each motor. Peak power is assumed to occur at the motor stall torque, when the maximum load is lifted in the fully extended position. Peak power of each motor is labeled MotorL1A1PeakPower, MotorL2A1PeakPower, MotorL3A1PeakPower, and MotorL4A1PeakPower, respectively.

The second group of outputs are the no load speeds of each motor. This is the speed, in [rad/sec], at which the motor could spin without any load applied. These are labeled L1A1SpeedMaxEff, L2A1SpeedMaxEff, L3A1SpeedMaxEff, and L4A1SpeedMaxEff, respectively.

The third group of outputs contain the masses [kg] of each motor and gearbox combination. As shown in the subsequent section, the motors are sized based off of a curve fit relating torque to mass. The gearboxes are then sized based off of a curve fit relating the gearbox ratio and the mass of the motor to the mass of the gearbox. The motor mass is added to

the gearbox mass and output as L1A1TotalMotorMass, L2A1TotalMotorMass, L3A1TotalMotorMass, and L4A1TotalMotorMass, respectively.

Masses of the motors and gearboxes are followed by their respective maximum output torques in units of [N-m]. The torque of each motor and gearbox combination is labeled L1A1torque, L2A1torque, L3A1torque, and L4A1torque, respectively. These torques are obtained from moment balance equations during a horizontal lift. Derivation is detailed in the next section.

Each segment of the manipulator is assumed to be hollow cylindrical tubes. The inner radius of each segment is a user-defined input. The outer radius is calculated through an iterative loop which increases the wall thickness until it satisfies a maximum stress condition. This loop is described in more detail in the next section. The outer segment radii are labeled L1A1radius, L2A1radius, and L3A1radius, respectively. These radii have units of [m].

Finally, the Manipulator subsystem sums the total number of user-defined degrees of freedom that are actually used by the manipulator. This output is labeled totalDOF and is dimensionless. While the user can specify three degrees of freedom at motor L4A1, the user may not have defined enough segment lengths to actually utilize those degrees of freedom. This summation provides a foolproof check to determine the number of defined degrees of freedom that the manipulator actually has.

The four red lines splitting from the path of the bus creator within the ‘black box’ subsystem to the bus selector, shown in Figure 9, indicates that the outputs from this subsystem are then used as inputs to four other subsystems. These subsystems are Batteries, Total Vehicle Dimensions, Total Vehicle Mass, and Manipulator Capabilities.

3.1.2 Model Derivation

From a high level, the manipulator subsystem follows a simple creation path. As shown in Figure 11, all of the manipulator components are sized beginning with the wrist motor (L1A1), followed by the elbow motor (L2A1), and sequentially followed by the forearm segment (L1A1). If the user defines only one segment link, then the subsystem outputs zeros for all of the remaining parameters. If the user defines two or more segment links, then the subsystem sizes the shoulder motor (L3A1) followed by the humeral segment (L2A1). Again, if the user defines three segment links, then the subsystem continues calculating the torso motor (L4A1) and finally the torso segment (L3A1).

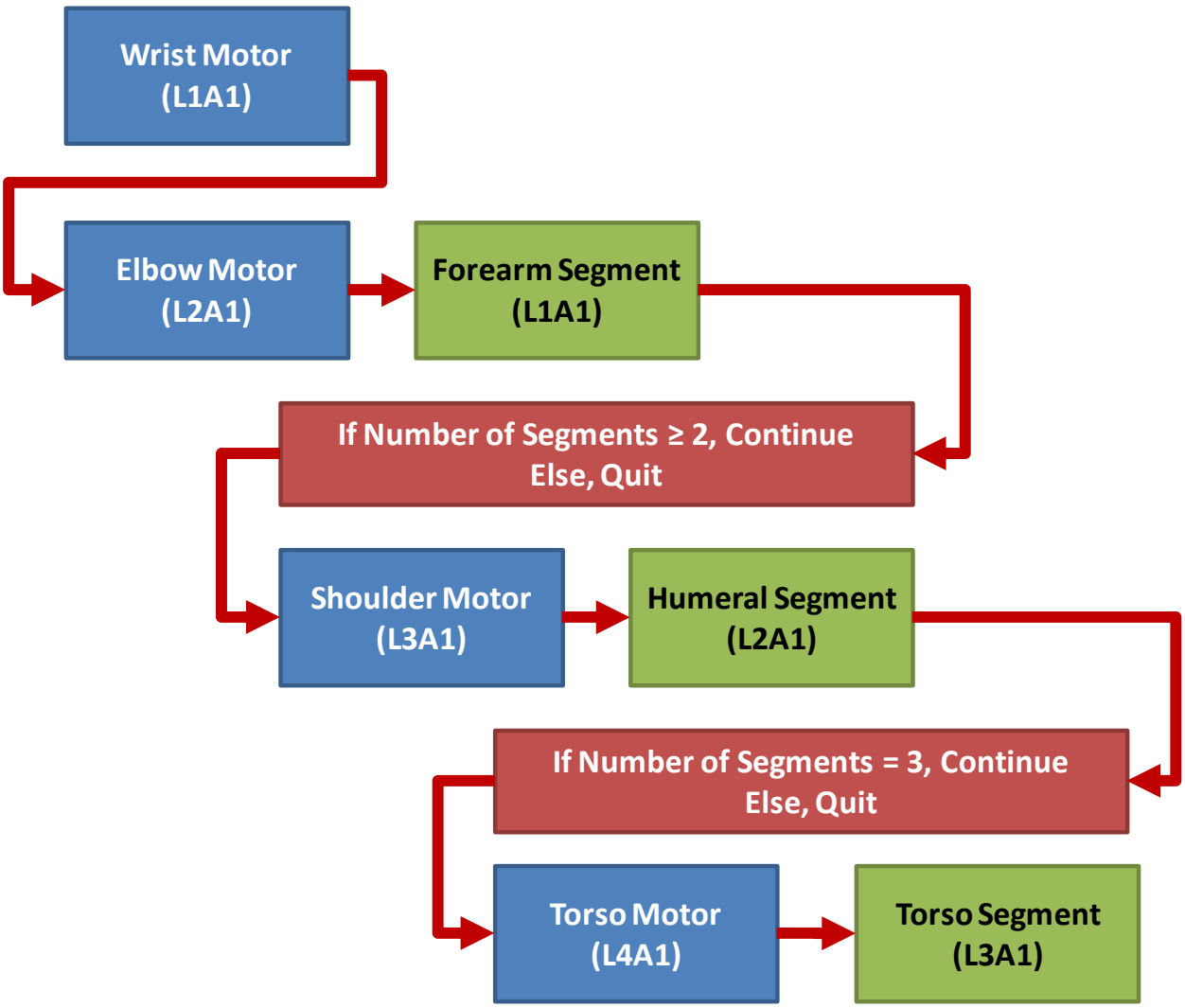

Figure 11: Manipulator Block Diagram

Sizing a motor first requires calculation of how much torque the motor must be able to support. This is done by isolating the motor in question with any links between the lifting force and the motor to be sized, from the rest of the system. This isolation is shown in Figure 12. The motors are sized for the most difficult static lifting situation: when the arm is stretched horizontally to lift mass FLift, which has units of [kg]. From this diagram, and the assumption that the motors are point masses and that the segment lengths are uniform along their lengths,

thus placing their centers of gravity at half of their lengths, Equation 3 can be derived from a moment balance about the point mass motor.

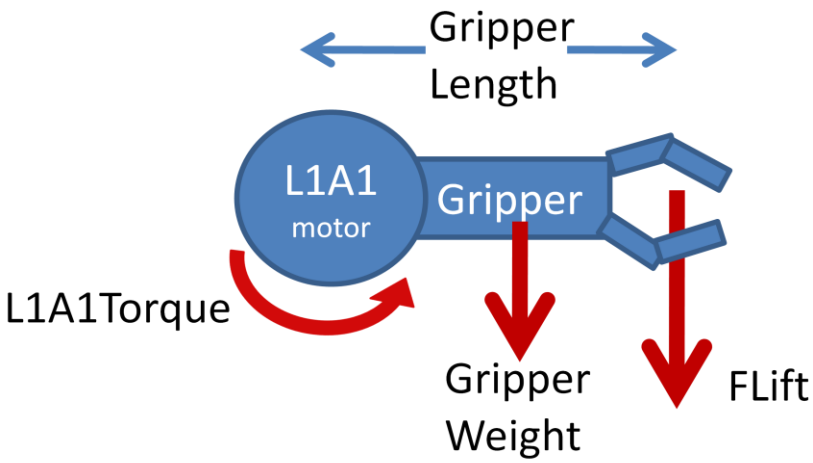

Figure 12: L1A1 Motor Sizing Diagram

$$L1A1TorqueTotal = \frac{GripperLength}{2} * (GripperMass * g) + GripperLength * (FLift * g) \quad (3)$$

Once the required torque for this motor is determined, all remaining parameters for this motor are calculated. First, the motor torque apart from the gearbox is determined from Equation 4. The gearboxes in the manipulator are used to slow down the motors and increase their torque output. Thus, in Equation 4, the total required lifting capacity is divided by the gearbox ratio to determine the portion of torque that the motor must be sized to lift.

$$L1A1TorqueMotor = \frac{L1A1TorqueTotal}{GearboxL1A1Ratio} \quad (4)$$

All of the remaining parameters are calculated using best-fit-curves from vendor data for several electric motors [26]. The first parameter to be calculated from these fit curves is the efficiency of the gearbox, shown in Equation 5. Once the gearbox efficiency is calculated the required torque of the motor is recalculated to make up for this efficiency loss, via Equation 6.

$$GearboxL1A1Efficiency = 0.931 - 0.06 * \log(GearboxL1A1Ratio) \quad (5)$$

$$L1A1TorqueMotor = L1A1TorqueMotor * GearboxL1A1Efficiency \quad (6)$$

The mass of the motor is calculated using Equation 7. This equation was created by fitting an equation to the vendor data in Figure 13. In this figure, the curve fit is shown in relation to the vendor data. A curve fit is used to relate the gearbox ratio and the mass of the motor to the mass of the gearbox as shown in Equation 8. The mass of the gearbox is then added to the mass of the motor and output as L1A1TotalMotorMass.

$$MotorL1A1Mass = 0.7309 * L1A1TorqueMotor^{0.4834} \quad (7)$$

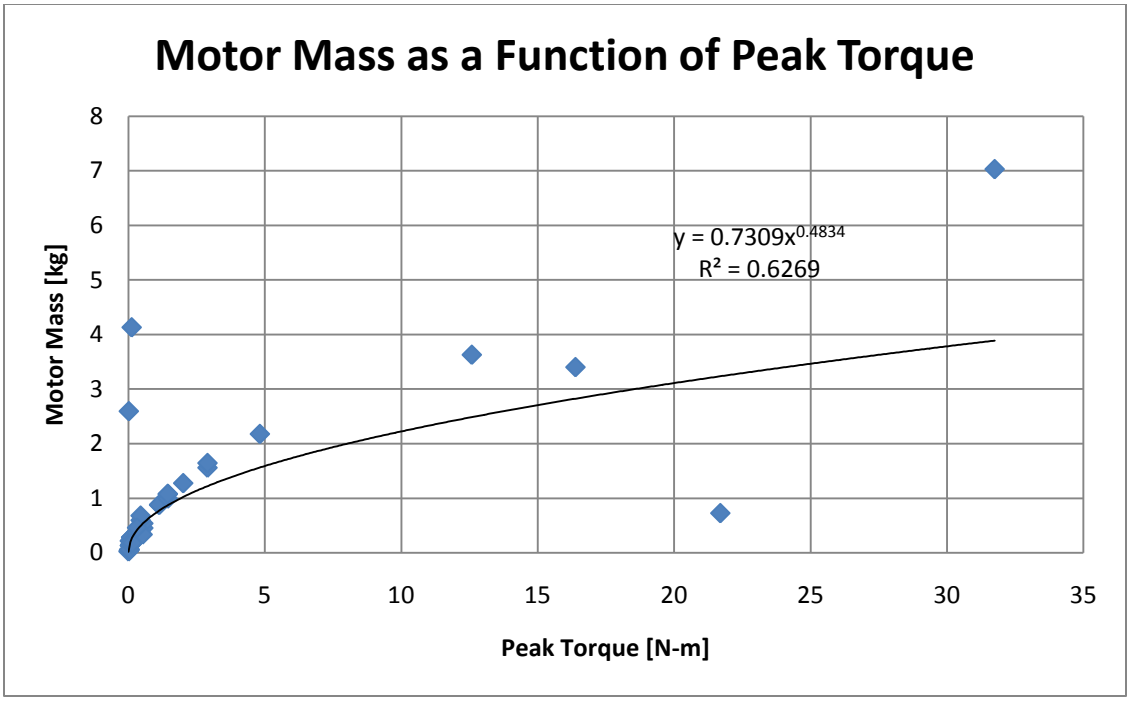

Figure 13: Motor Mass as a Function of Peak Torque

$$GearboxL1A1Mass = MotorL1A1Mass * (0.004 * GearboxL1A1Ratio + 0.560) \quad (8)$$

Motor speed at maximum efficiency is then calculated as a function of the peak motor torque defined in Equation 6, using Equation 9. This equation's fit to vendor data is shown in Figure 14.

$$MotorL1A1SpeedMaxEff = 641.55 * e^{-0.349 * L1A1Torque} \quad (9)$$

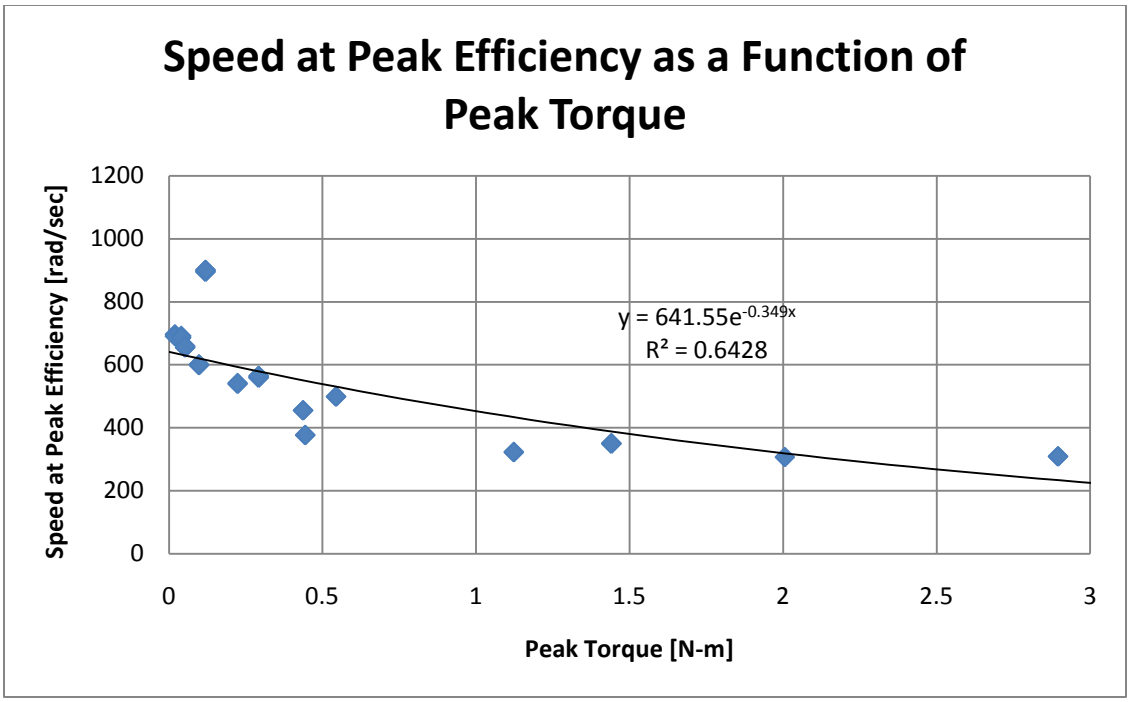

Figure 14: Speed at Maximum Efficiency as a Function of Peak Torque

Finally, the maximum power draw of the motor is calculated using Equation 10. This curve fit is compared to vendor data in Figure 15.

$$MotorL1A1PowerStall = -1.8339 * L1A1Torque^2 + 143.65 * L1A1Torque + 16.494 \quad (10)$$

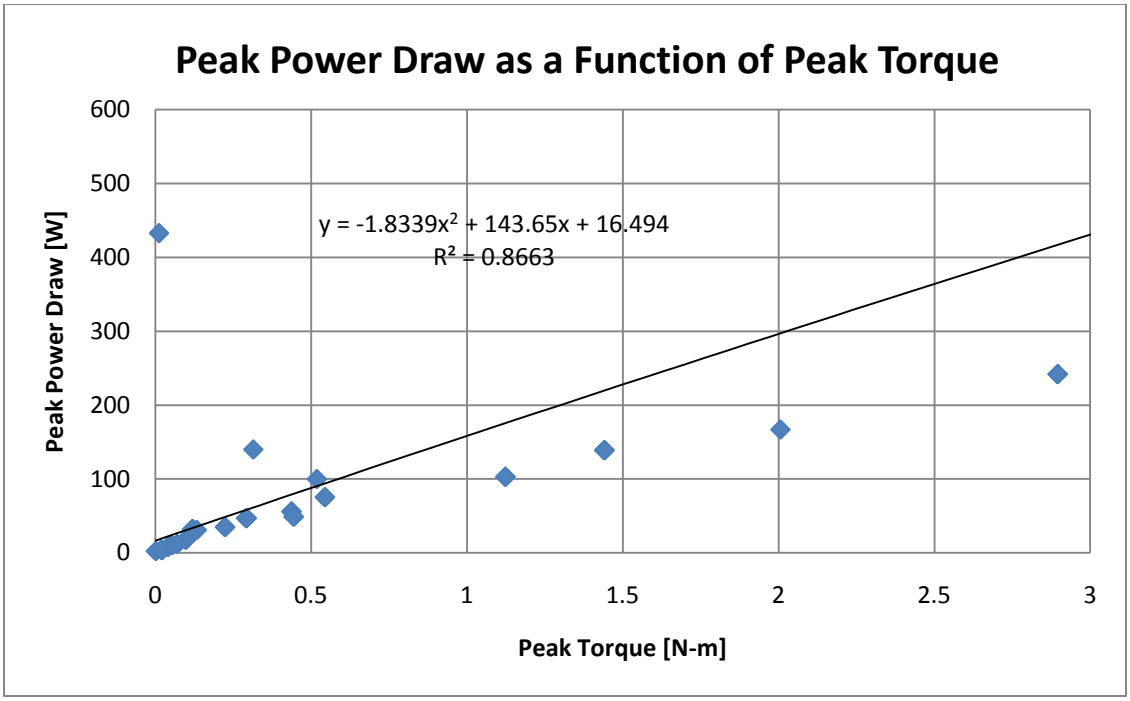

Figure 15: Peak Power Draw as a Function of Peak Torque

Once all of the parameters have been determined for MotorL1A1, the next motor, MotorL2A1, can then be sized. However, the mass of segment L1A1 factors into the lifting equation for Motor L2A1. The length of the segment, inner radius of the segment, and material density are all user-defined inputs to the model. Thus, segment L1A1 can be accurately sized by iteratively increasing the segment's thickness. The segment's thickness is initially defined as 0.1 [cm]. The value for segment thickness will be iteratively increased until the segment will not break. The mass of the segment is calculated as density multiplied by volume, shown in Equation 11.

$$segmentL1A1mass = armDensity * L1A1Length * \pi * [(innerArmRadiusL1A1 + thicknessL1A1)^2 - innerArmRadiusL1A1^2] \quad (11)$$

MotorL2A1 is sized similarly to MotorL1A1 but calculated using a slightly revised lifting equation. The revised lifting equation is also derived from a simple moment balance but derived from the free body diagram in Figure 16, and shown in Equation 12.

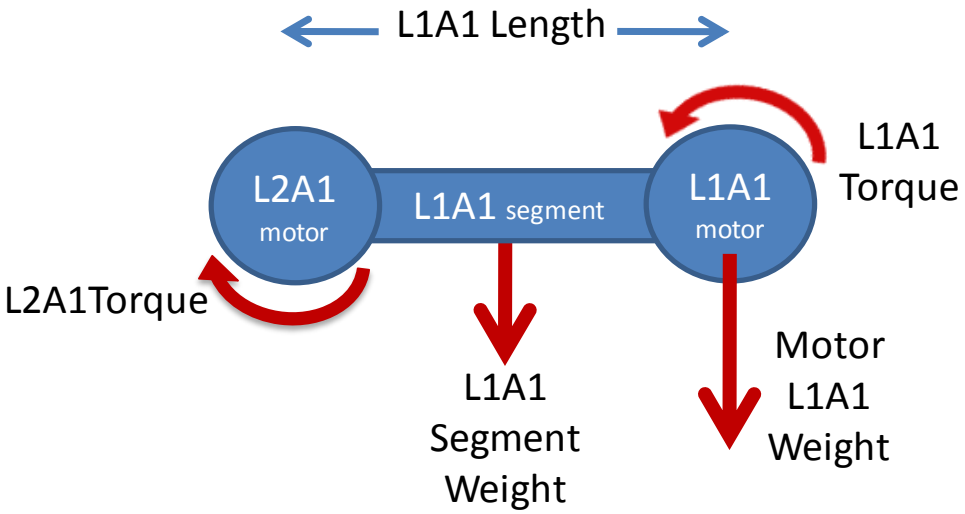

Figure 16: L2A1 Motor Sizing Diagram

$$L2A1TorqueTotal = \frac{L1A1Length}{2} * (segmentL1A1Mass * g) * nDOFL1A1 + L1A1Length * TotalMotorL1A1Mass * g + L1A1TorqueTotal \quad (12)$$

Once MotorL2A1Torque is known, GearboxL2A1Efficiency is calculated using Equation 5, and MotorL2A1Torque is recalculated as it was for MotorL1A1Torque using Equation 6. MotorL2A1mass and GearboxL2A1mass are calculated using Equation 7 and Equation 8, respectively.

Next a check is performed to make sure that the segment does not break while lifting the user-defined capacity, FLift. If FLiftMax, as defined in Equation 13, is greater than FLift multiplied by a safety factor of two, then the arm yields. Equation 13 is derived from a stress

calculation. If the segment yields, then the thickness of the arm segment is increased by 0.1cm, and the analysis for sizing the segment and motor is repeated.

$$FLiftL1A1Max = \frac{1}{L1A1Length+GripperLength} * \left[\frac{\{armStress*\pi*[(innerArmRadiusL1A1+thicknessL1A1)^4 - innerArmRadiusL1A1^4]\}}{4*(innerArmRadiusL1A1+thicknessL1A1)} - GripperMass*g*L1A1Length+GripperLength2-TotalMotorL1A1Mass*g*L1A1Length-L1A1Length*L1A1mass*g2 \right. \quad (13)$$

Once an appropriate segment thickness is determined, the maximum power draw and speed at peak efficiency are determined using the same fit curves as for the wrist motor shown in Equation 9 and Equation 10.

Per Figure 11, if the user has specified two or more segments, then the same analysis is performed for the next motor and segment link. If the user has specified only one segment, then the model outputs null values for all of the remaining parameters.

Once the entire manipulator has been appropriately sized, all of the user-defined degrees of freedom are summed together and output from the subsystem as totalDOF. A check is performed to ensure that only the degrees of freedom that are actually used are summed together. For example, if the user specifies only one segment link but specified degrees of freedom at the shoulder and torso motors, the model ignores those unused degrees of freedom.

Likewise, the masses of all of the motors, gearboxes and segments are summed together and output as TotalManipulatorMass. The power draw of each motor is summed together along

with a user-defined value for the end-effector power draw and control electronics and output as MaxManipulatorPowerDraw.

3.2 Battery Modeling

3.2.1 Subsystem Overview

A high level view of the Battery subsystem is shown in Figure 17. Like the Manipulator subsystem, the inputs to the Battery subsystem are the light blue boxes to the left of the subsystem 'black box', which is shaded green. The light blue coloring of the input boxes indicates that the parameter is a user-defined input. The Battery subsystem relies almost entirely on user-defined inputs. These inputs include batteryType [NiCad, niH2, PbAcid, NiMH, LiIon, or BB2590U], busVoltage [V], and batteryCapacity [Amp-hrs]. Additionally, the subsystem also relies on one output from the Manipulator subsystem. It can be seen that this input is from another subsystem because a bus selector is used to select the appropriate output parameter from its originating subsystem. In addition to tracing the path of this variable from the Manipulator subsystem from within the Simulink® diagram, it can be seen that this input is output from the Manipulator subsystem from its red foreground color. Each subsystem has its own unique color to indicate the subsystem of origin when the outputs are used in other subsystems.

Outputs of the Battery subsystem include the dimensionless Peukert number, which is used to calculate runtime in the Power Requirements subsystem [26]. Dimensions of the batteries are output as batteryLength, batteryWidth, and batteryHeight each with units of [m]. The volume of the battery is output as batteryVolume [m^3]. The mass of the battery is labeled batteryMass [kg], and the maximum power output is batteryPower [W]. If an off-the-shelf BB2590U battery (batteryType = 6) is specified, then the subsystem calculates the required

number of batteries as numberOfBatteries. If a non-standard battery is specified (i.e., batteryType = 1 through 5) then the subsystem always outputs a required number of batteries equal to one with the mass and capacity determined by the subsystem. Because it is harmful to many battery chemistries to completely discharge a battery, the subsystem outputs the battery capacity factoring in a safe depth of discharge capacity as batteryCapacityWithDOD with units of [Amp-hrs]. Finally, the subsystem outputs a flag to indicate whether or not the batteries are capable of supplying the manipulator with its peak power demand. This flag is labeled manipulatorPowerFlag and outputs a 0 if the batteries are able to meet the peak power demands or outputs a 1 if the batteries are unable to meet the peak power demands.

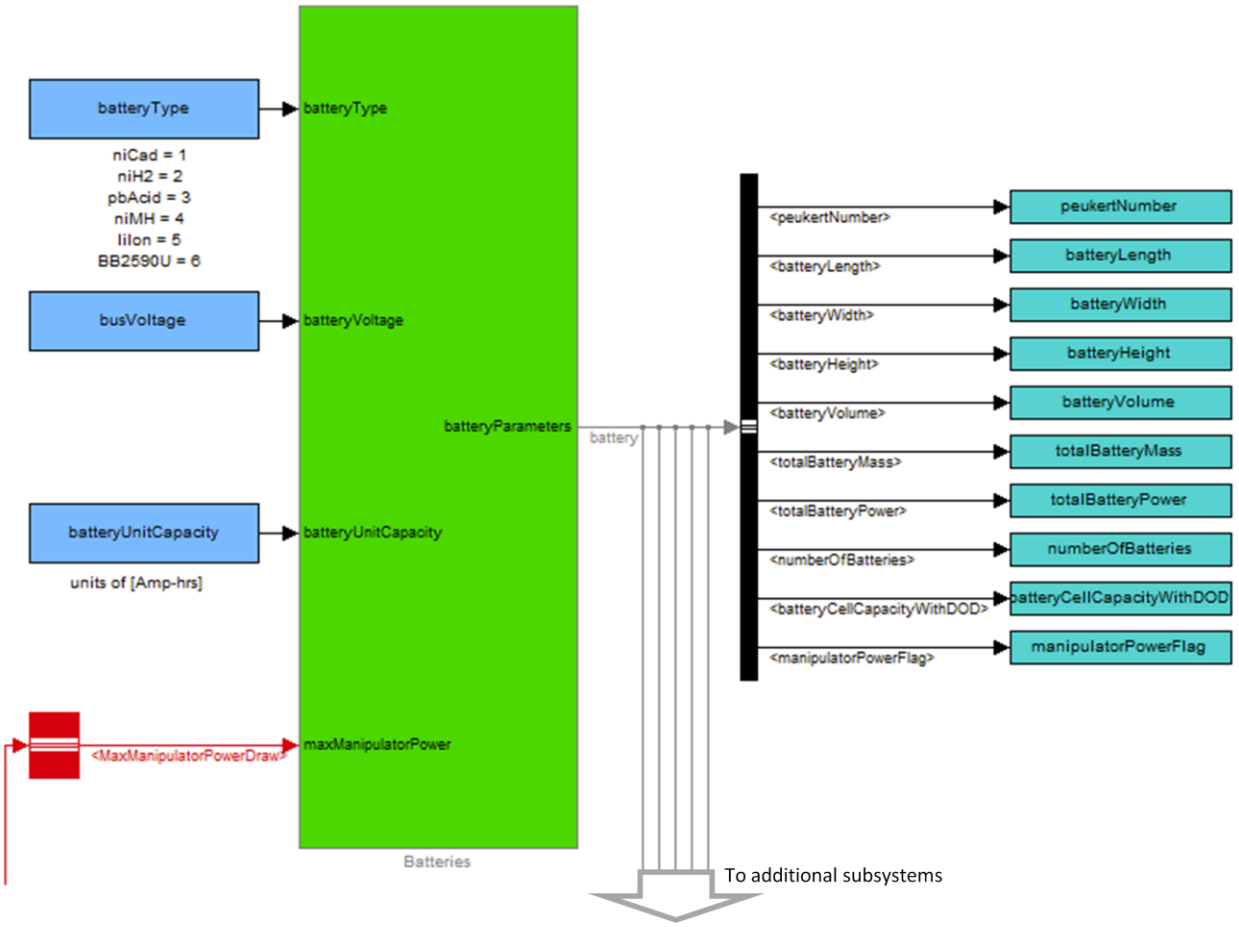

Figure 17: Simulink® Battery Subsystem

Just as in the Manipulator subsystem, it can be seen from Figure 17, that the outputs from the Batteries subsystem are used as inputs for five other subsystems. Again, this is shown by the five gray colored signal buses branching from the subsystem output. These five subsystems include the Motor Controller, Drive Motor, Power Requirements, Endurance, and Total Vehicle Mass subsystems.

3.2.2 Model Derivation

As mentioned in Section 3.2.1, there are three input parameters that must be known in order to size a battery. The model-defined MaxManipulatorPowerDraw input is not critical to sizing of the battery, but rather it is used to determine if a flag should be output if the user does not input a large enough battery capacity to meet the peak power demands of the manipulator. From these three inputs, this subsystem can output all of the necessary power parameters required to output a robot to the tradespace.

The first action performed within the Battery subsystem is selection of five chemistry-specific battery constants. These constants and their associated units can be found in Table 5 and include the Peukert Number, specific energy, specific density, cell voltage and maximum depth of discharge percentage. The subsystem determines the constants to pull from Table 5 based on the user-defined battery type. The Peukert Number is directly output from this subsystem to be used later in the Power Requirements subsystem. The other constants are used for subsequent calculations within the Batteries subsystem.

Table 5: Chemistry Specific Battery Constants

Number	Battery Type	Peukert Number	Specific Energy	Specific Density	Cell Voltage	Depth of Discharge
		[ ]	[W-hrs/kg]	[W-hr/m^3]	[V]	[%]
1	NiCad	1.2	45	105000	1.2	100
2	NiH2	1.2	48	150000	1.5	80
3	PbAcid	1.4	40	100000	2.1	80
4	NiMH	1.2	70	175000	1.2	80
5	LiIon	1.1	102.5	200000	3.6	80
6	BB2590U	1.1	127.5	202160	3.6	80

After determination of the chemistry-specific battery constants, the subsystem splits into two directions. The subsystem takes one path if a custom battery (battery types 1 though 5) is selected and the other path if an off-the-shelf battery such as the BB2590U (battery type 6) is

selected. The model can easily be expanded to include other battery chemistries or off-the-shelf batteries.

Custom Battery:

The volume of a battery is a function of its capacity, voltage, and battery density. Battery density is a constant based on battery chemistry. As previously mentioned, capacity and voltage are user-defined model inputs. The volume of the custom battery is calculated using Equation 14.

$$batteryVolume = \frac{capacityOfBattery * voltageOfBattery}{batteryDensity} \quad (14)$$

The mass of a battery is a function of its capacity, voltage and battery specific energy. Like battery density, specific energy is a constant based on battery chemistry; additionally, battery capacity and voltage are user-defined model inputs. The mass of the battery is calculated using Equation 15.

$$batteryMass = \frac{capacityOfBattery * voltageOfBattery}{batterySpecificEnergy} \quad (15)$$

The maximum power that a battery can supply is a function of mass and specific energy. The mass of the battery is calculated in Equation 15, and it is assumed that specific energy is a constant based on battery chemistry; therefore, the maximum power available from the battery is calculated using Equation 16.

$$batteryPower = batteryMass * batterySpecificEnergy \quad (16)$$

Finally, the physical dimensions of the battery are calculated. While custom batteries can take a variety of shapes depending on how individual cells are manufactured, the custom batteries in this analysis are assumed to be a rectangular prism. The height of the battery is held constant at 0.125 [m], the same as the BB2590U. The height is held constant in order to make trade studies slightly easier during post processing. The width of the battery is calculated as a function of volume and height as shown in Equation 17. Finally, the length of the battery is calculated as a function of width as shown in Equation 18.

$$batteryWidth = \sqrt{\frac{batteryVolume}{batteryHeight/1.8}} \quad (17)$$

$$batteryLength = 1.8 * batteryWidth \quad (18)$$

Off-The-Shelf Battery (BB2590U):

If a BB2590U battery type is selected then all of the parameters that must be calculated for a custom battery are already known. The BB2590U is commonly used in military and civilian systems and shares a common size with several other off-the-shelf batteries. The length, width, and height of one BB2590U is 0.11176 [m], 0.06223 [m], and 0.127 [m], respectively. The mass of each battery is fixed at 1.4 [kg]. The required number of batteries is chosen by Equation 19. In Equation 19, the Matlab® function 'ceil' rounds up to the nearest integer. Because a BB2590U has eight cells per battery, this parameter is fixed at eight.

$$numberOfBatteries = ceil\left[\frac{voltageBattery}{voltagePerCell * numberCellsPerBattery}\right] \quad (19)$$

The maximum battery power available is calculated the same way as for a custom battery, using Equation 16, however, it is scaled by the number of BB2590U batteries in the system. Finally, the total battery mass is merely the predefined 1.4 [kg] mass of one BB2590U multiplied by the number of batteries in the system.

3.3 Motor Controller Modeling

3.3.1 Subsystem Overview

Sequentially, the next subsystem processed after the Batteries subsystem sizes the drive motors' motor controller. The Simulink® 'black box' for this subsystem is shown in Figure 18. This subsystem contains three user-defined inputs including auxiliary power draw [W], which is the average power draw of any on-board auxiliary systems; the number of wheels driven by a motor; and the drive motor configuration, which determines how the drive motors connect to each driven wheel. The drive motor configuration cases are (1) a single motor to drive all driven wheels, (2) one drive motor per driven axle, and (3) one drive motor per wheel.

The first output from the subsystem is a binary flag that triggers a value of one if the batteries are unable to provide enough power to run an electric motor. If a motor is incapable of being sized, then a zero is output. This minimum power output value is set as 3 [W]. This value was chosen because it is not only the smallest value that would work in the best fit curves within the Drive Motor subsystem but also the smallest power value found in the electric motor vendor data. Additionally, mass [m], length [m], width [m], height [m], and efficiency [%] are output from the subsystem.

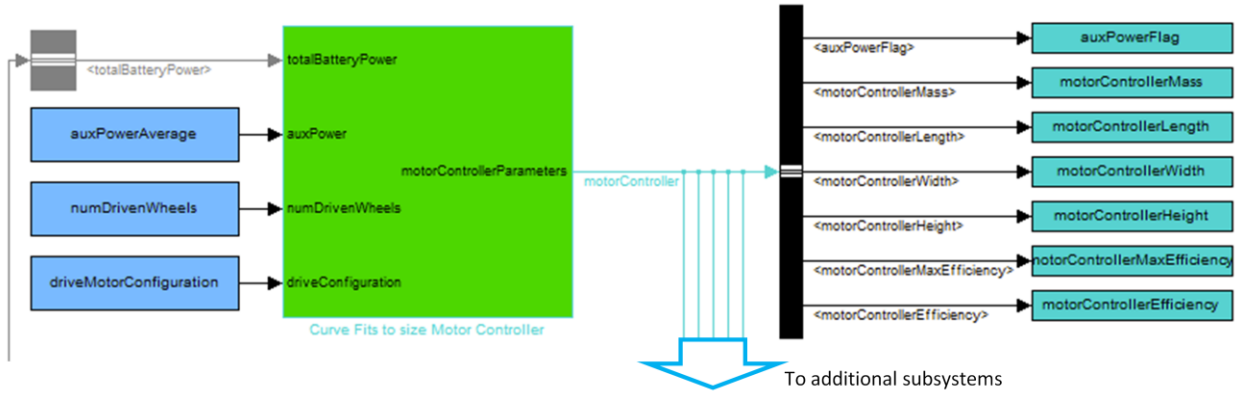

Figure 18: Simulink® Motor Controller Subsystem

It can be seen in Figure 18, from the five cyan colored signal buses branching from the output of the Motor Controller subsystem that the outputs from this subsystem are used as inputs to five other subsystems. These five subsystems include the Drive Motor, Chassis Dimensions, Power Requirements, Endurance, and Total Vehicle Mass subsystems.

3.3.2 Model Derivation

The motor controller is sized based off of experimental data best fit curves. The outputs are all scaled based on the total power available at each wheel, MCPower, as shown in Equation 20. In Equation 20, the total battery power [W] is calculated in the Battery subsystem and the average auxiliary power draw [W] is a user-defined input. Average auxiliary power draw accounts for the power needed for sensors, computers, and other robot electronics. The dimensionless number of motors is determined based on the number of driven wheels and the drive motor configuration user-defined inputs. If the drive motor configuration is equal to one, then the number of motors is one. If the configuration parameter is equal to two, which is the "one motor per axle" configuration, then the number of drive motors is equal to the number of driven wheels divided by two. If the third drive motor configuration is selected, which is the

"one motor per wheel" configuration, then the number of motors is equal to the number of driven wheels.

$$MCPower = \frac{totalBatteryPower - auxPowerAverage}{numberOfMotors} \quad (20)$$

Four outputs from this subsystem are computed from best fit curves created from available vendor data. These outputs are motor controller mass [kg], length [m], width [m], and height [m]. The best fit curves are shown in Equation 21, Equation 22, Equation 23, Equation 24, respectively. The maximum motor controller efficiency is set equal to a constant 94.798% and the average motor controller efficiency is set to a constant 92.235%. Values for both maximum and average motor controller efficiency located in vendor data were consistently found to have little deviation. Therefore, these two parameters were determined to be the average of the values found in the vendor data.

$$motorControllerMass = 0.0002 * MCPower + 0.0042 \quad (21)$$

$$motorControllerLength = 0.0537 + 0.00003 * MCPower - 9E(-10) * MCPower^2 \quad (22)$$

$$motorControllerWidth = 0.0084 * MCPower \quad (23)$$

$$motorControllerHeight = 0.004 * MCPower^{0.3083} \quad (24)$$

3.4 Drive Motor Modeling

3.4.1 Subsystem Overview

Sequentially following the Motor Controller subsystem is the Drive Motor subsystem. As shown in Figure 19, this subsystem is preceded by the Motor Controller and Battery subsystems because it depends on outputs from both subsystems. Sizing a drive motor depends on the total battery power available which is calculated in the Battery subsystem. It also depends on the maximum motor controller efficiency and knowledge of whether or not the auxPowerFlag was output in the Motor Controller subsystem. As mentioned in Section 2.5, the gearbox within the Drive Motor subsystem depends on maximum vehicle velocity which must be iteratively solved for between subsystems with the use of a feedback loop. Additionally, the subsystem depends on five user-defined inputs. These inputs are the number of driven wheels, wheel diameter [m], drive motor configuration, bus voltage [V], and auxiliary power draw [W]. Wheel diameter is the only user-defined input used for the first time within the model.

There are 19 parameters output from this subsystem. These parameters are the motor constant K, motor mass [kg], motor length [m], motor diameter [m], no load current [amps], armature resistance [ohms], motor reference voltage [V], motor stall torque [N-m], maximum motor efficiency [%], motor no load speed [rad/s], gearbox efficiency [%], combined motor and gearbox efficiency [%], maximum output torque of the motor and gearbox [N-m], maximum output steady state torque [N-m], gearbox ratio, number of motors as defined by the number of driven wheels and drive motor configuration, gearbox mass [kg], and finally the total mass of the motor and gearbox combination [kg].

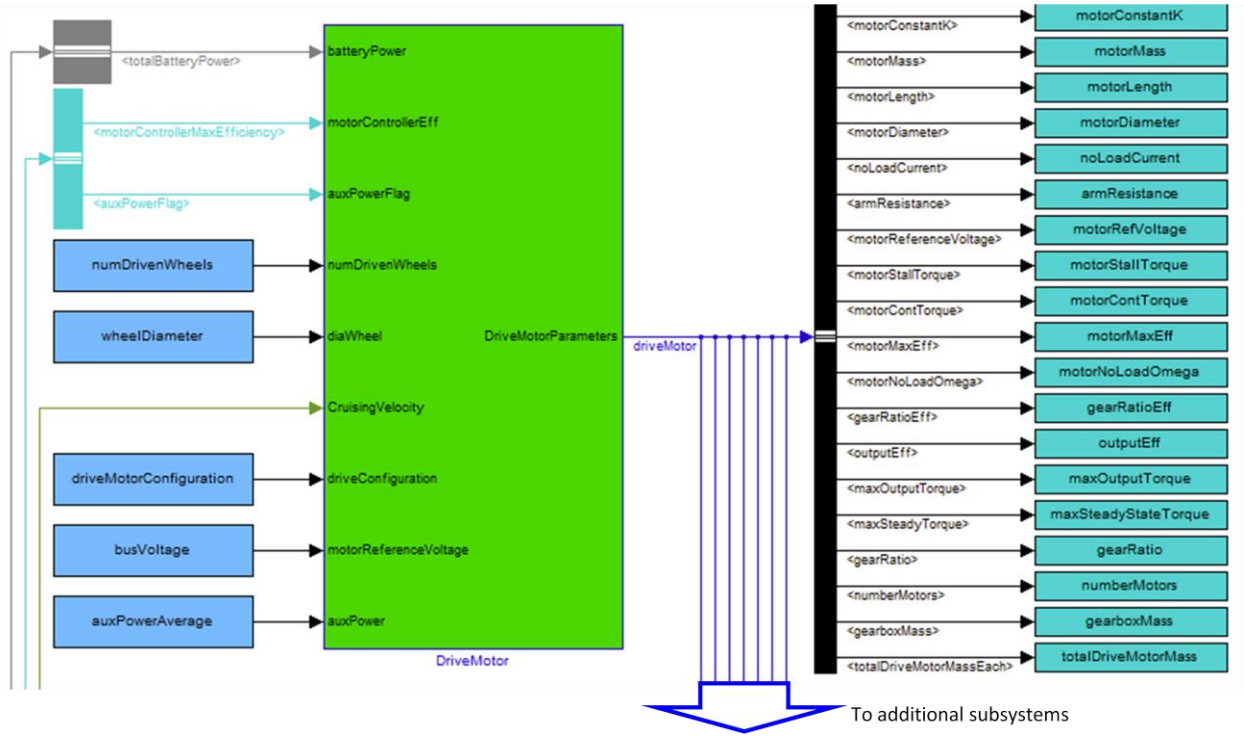

Figure 19: Simulink® Drive Motor Subsystem

As shown in Figure 19, there are seven other subsystems that depend on outputs from the Drive Motor subsystem. These dependent subsystems are Chassis Dimensions, Power Requirements, Endurance, Total Vehicle Dimensions, Total Vehicle Mass, Functional Capabilities, and Manipulator Capabilities.

3.4.2 Model Derivation

Many of the drive motor parameters are calculated based on best fit curves scaled on the available power supplied to each motor, which, as shown in Equation 20, is a function of available power from the battery, the number of driven wheels, and the drive motor configuration. Additionally, Equation 20 is reduced by an efficiency loss from the motor controller. There are ten parameters that are scaled based on curve fits. These parameters are

the motor armature resistance, no load current, motor constant K, no load speed, motor length, motor diameter, stall torque, continuous torque, speed at maximum efficiency, and mass. All ten of these equations are either outputs from the model or used in further calculations. These curve fits were created from the same vendor data as the electric motors in the Manipulator subsystem. The curve fits are given in Equation 25 through Equation 34, respectively.

$$armatureResistance = 41.245 * motorPower^{-1.005} \tag{25}$$

$$noLoadCurrent = 5E(-6) * motorPower^2 + 0.0028 * motorPower + 0.1589 \tag{26}$$

$$motorConstantK = -1E(-9) * motorPower^2 + 3E(-5) * motorPower + 0.0323 \tag{27}$$

$$noLoadOmega = 929.49 * motorPower^{-0.116} \tag{28}$$

$$motorLength = 0.036806 * motorPower^{0.2169} \tag{29}$$

$$motorDiameter = 0.01657 * motorPower^{0.3042} \tag{30}$$

$$stallTorque = 3E(-8) * motorPower^2 + 0.013 * motorPower + 0.3578 \tag{31}$$

$$contTorque = 4E(-5) * motorPower^2 + 0.0024 * motorPower - 0.0134 \tag{32}$$

$$maxEffOmega = 740.29e^{-0.014 * motorPower} \tag{33}$$

$$motor\ mass = 0.0386 * motorPower^{0.7098} \quad (34)$$

Once the previous ten parameters have been evaluated, the gearbox is sized based on the necessary wheel speed. The rotational speed of the wheels, in units of [rad/s], required for the vehicle to travel a given velocity is shown in Equation 35. As shown in Equation 36, the gearbox is sized to match the speed of the motor at peak efficiency to the necessary rotational speed. The drive motor gearbox mass is calculated using the same fit curve used in the Manipulator subsystem shown in Equation 8. The mass of the gearbox is then added to the drive motor mass to obtain the total drive motor and gearbox mass.

$$continuousRotationalSpeed = \frac{maxCruisingVelocity}{wheelDiameter/2} \quad (35)$$

$$gearboxRatio = \frac{motorMaxEffOmega}{continuousRotationalSpeed} \quad (36)$$

Next, the maximum efficiency of the drive motor is determined by first calculating the loss resistance, R_h, in Equation 37 having units of [ohms]. Armature resistance and loss resistance are then used in Equation 38 to calculate the dimensionless motor constant M. The voltage drop across the brushes is calculated using Equation 39. Finally, the maximum motor efficiency is calculated as a function of reference voltage, which is set equal to the user-defined bus voltage, voltage drop across the brushes, and the motor constant M using Equation 40.

$$R_h = \frac{motorConstantK * noLoadSpeed}{noLoadCurrent} \tag{37}$$

$$M = \sqrt{\frac{R_a + R_h}{R_a}} \tag{38}$$

$$V_{brush} = RefVoltage - R_a * noLoadCurrent - motorConstantK * noLoadSpeed \tag{39}$$

$$motorMaxEff = \frac{ReferenceVoltage - V_{brush}}{ReferenceVoltage} * \frac{M - 1}{M + 1} \tag{40}$$

Similarly to the gearboxes in the Manipulator subsystem, gearbox efficiency is calculated using Equation 5. The output efficiency is simply a combination of the drive motor efficiency multiplied by the gearbox efficiency. The drive motor torque is then adjusted to incorporate the gearbox ratio and efficiency loss as shown in Equation 41 and Equation 42.

$$maxOutputTorque = motorStallTorque * (gearboxRatio * gearboxEfficiency) \tag{41}$$

$$maxSteadyTorque = motorContTorque * (gearboxRatio * gearboxEfficiency) \tag{42}$$

3.5 Chassis Dimensions

3.5.1 Subsystem Overview

The Chassis Dimension subsystem is used to determine the necessary dimensions of the robot's chassis. These chassis dimensions are outputs of the subsystem and include chassis width [m] and chassis height [m], as well as the total vehicle length [m].

As shown in Figure 20, the first three user-defined inputs to the subsystem are allotted dimensions for payload: payload length [m], payload width [m], and payload height [m]. This payload can be used to store computers, sensors, communications systems, etc. The Chassis Dimensions subsystem depends on two user-defined inputs that were first used in the Drive Motor and Motor Controller subsystems, respectively. These two inputs are wheel diameter [m] and the drive motor configuration. The final user-defined input is the desired additional front length [m]. This input is necessary to design a robot such as the BomBot because one of its batteries is positioned in front of the front wheels and its other battery is positions behind its rear wheels.

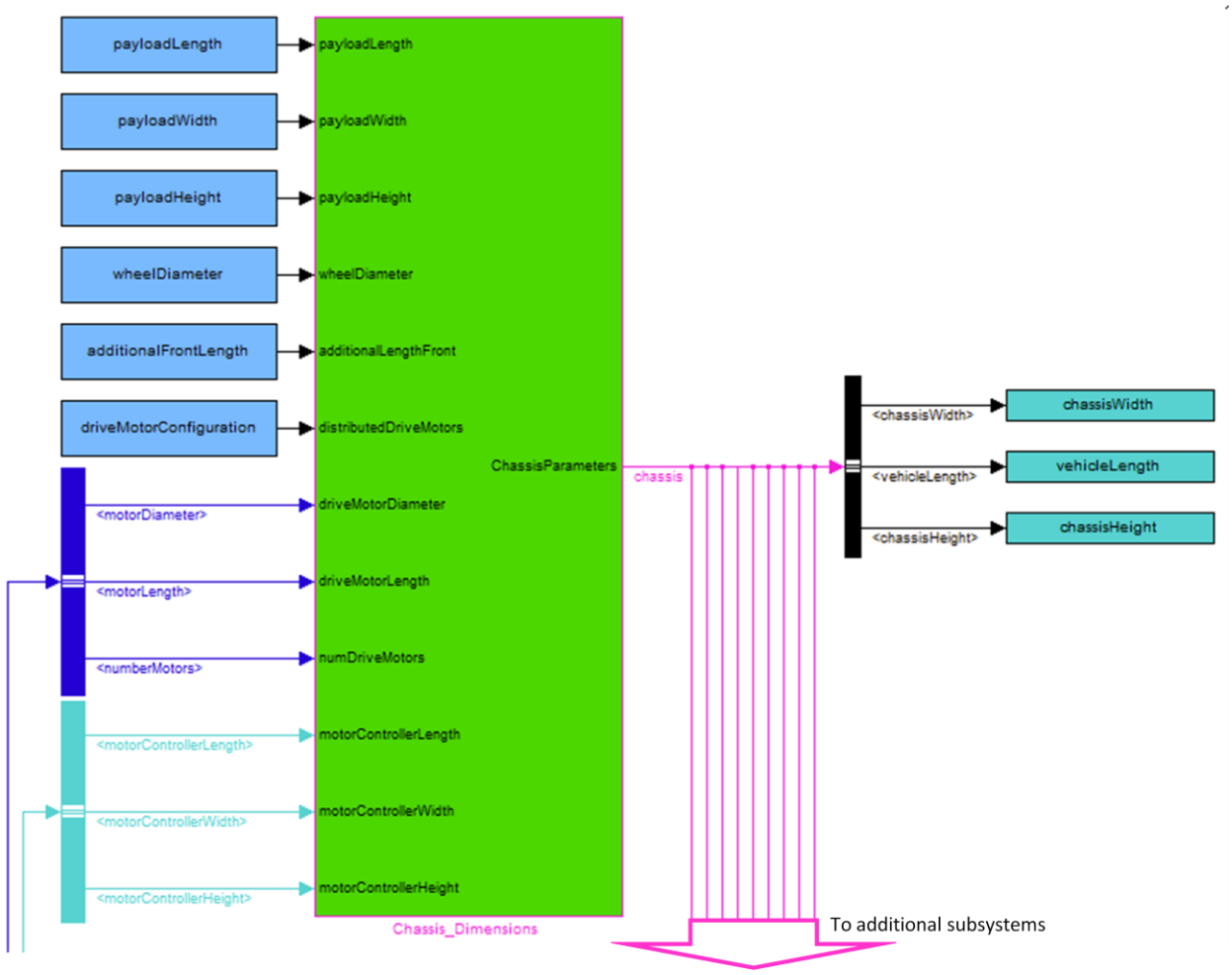

Figure 20: Simulink® Chassis Dimensions Subsystem

As seen in Figure 20, the outputs from this subsystem are used as inputs to eight other subsystems. These subsystems are the Structure, Wheels, Tracks, Power Requirements, Total Vehicle Dimensions, Functional Capabilities and Manipulator Capabilities subsystems.

3.5.2 Model Derivation

The first dimension to be calculated is the required length and width of the chassis to fit the drive motors and motor controllers. These dimensions are named PowerLength and PowerWidth, respectively. How these two dimensions are calculated depend on the user-defined

input for drive motor configuration whose input range is between one and three. The two equations used for these calculations are shown in Equation 43 and Equation 44.

$$if \begin{bmatrix} driveMotorConfiguration = 1 \\ driveMotorConfiguration = 2 \\ driveMotorConfiguration = 3 \end{bmatrix} then \;\; powerLength$$
$$= \begin{bmatrix} DMLength + MCLength \\ DMLength + MCLength \\ \max\left(DMDiameter + MCLength, \frac{numDriveMotors}{2} * DMDiameter * MCLength\right) \end{bmatrix} \quad (43)$$

$$if \begin{bmatrix} driveMotorConfiguration = 1 \\ driveMotorConfiguration = 2 \\ driveMotorConfiguration = 3 \end{bmatrix} then \;\; powerWidth$$
$$= \begin{bmatrix} DMLength + MCWidth \\ DMLength + MCWidth \\ DMLength * 2 + MCWidth \end{bmatrix} \quad (44)$$

Once PowerLength and PowerWidth are known, the chassis width [m] can be calculated. The chassis width is calculated as the maximum of powerWidth, payloadWidth, and guessWidth. GuessWidth is calculated using Equation 45. This equation first calculates an estimate for how long the chassis should be: payloadLength + powerLength. The equation then divides by an experimental length to width ratio of 1.65 to determine chassis width.

$$guessWidth = \frac{payloadLength + powerLength}{1.65} \quad (45)$$

VehicleLength [m] is calculated next. VehicleLength uses the same length to width experimental value as guessWidth. In Equation 46, 1.65*chassisWidth represents the required

length of the chassis. To obtain the overall length of the vehicle, wheel diameter is added to the chassis length because half of the wheel would hang over the front of the chassis and half over the rear of the chassis. The drive motor diameter is subtracted from this value in order to give the motors a location on the chassis to mount. The user-defined value for additionalFrontLength increases the overall length of the chassis if it is assigned a value.

$$vehicleLength = 1.65 * chassisWidth + wheelDiameter - driveMotorDiameter + additionalFrontLength \quad (46)$$

The height of the chassis is a much simpler calculation. It first determines powerHeight as the drive motor diameter plus the motor controller height. Chassis height is then determined as the maximum of the user-defined payloadHeight or powerHeight.

3.6 Chassis Structure

3.6.1 Model Overview

Once the Chassis Dimensions have been calculated, a suitable chassis structure is determined. The Chassis Structure subsystem does not attempt to provide an optimized design shape but rather determines the thickness of the rectangular platform on which to mount all of the vehicle components (refer to Figure 2 for a graphical representation of this chassis structure).

As shown in Figure 21, this subsystem depends on two outputs from the Chassis Dimensions subsystem, namely vehicle length [m] and chassis width [m]. It also depends on three user-defined inputs. The manipulator lifting capacity, FLift [kg] and wheel diameter [m] have already been used in previous subsystems. The internal payload mass [kg] is used for the

first time in this subsystem. This parameter is the mass of any computer, sensor, camera, etc. equipment stored on the vehicle. Finally, the subsystem depends on material properties for the desired material. The properties for aluminum have been hard-coded into the model. These properties include the elastic modulus [GPa], density [kg/m^3], yield stress [MPa], a safety factor, and the maximum allowable deflection [m]. These hard-coded material properties could easily be turned into user-defined inputs if trade studies involving various chassis materials are desired.

As shown in Figure 21, the first output from the Chassis Structure subsystem is the thickness of the chassis structure, labeled finalChassisHeight with units of [m]. The next output is the mass of the chassis, labeled chassisMass, with units of [kg]. The last output is the final chassis deflection, labeled finalDeflection, with units of [mm].

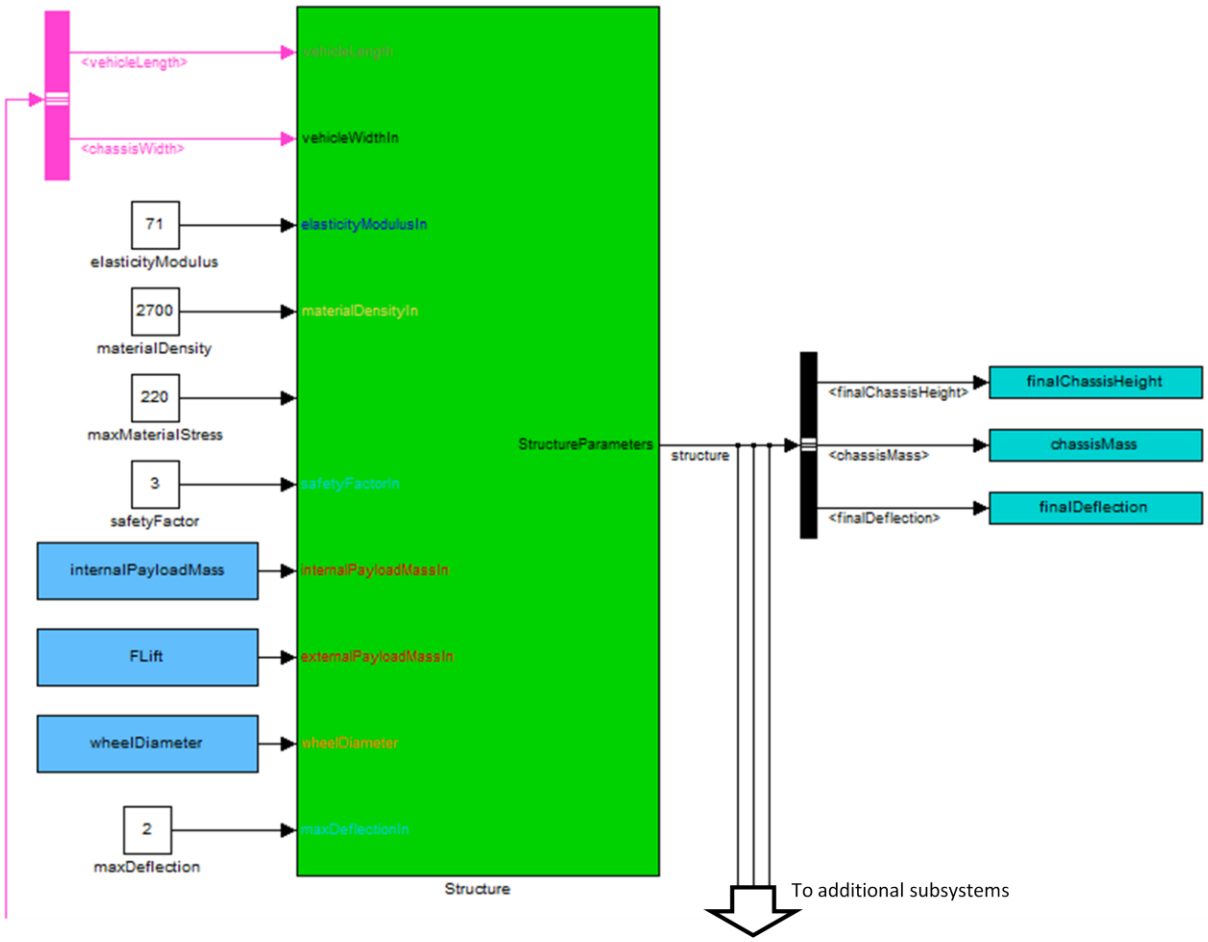

Figure 21: Simulink® Chassis Structure Subsystem

As shown in Figure 21, the outputs from this subsystem are used as inputs to three other subsystems. These subsystems include Endurance, Total Vehicle Dimensions, and Total Vehicle Mass.

3.6.2 Model Derivation

There are two limiting cases for which the thickness of the chassis must be designed. The first is the deflection-limited case, and the second is the stress-limited case. However, as mentioned in Section 2.5, the Chassis Structure subsystem contains a circular dependency on

chassis mass. Because the chassis structure has mass, it must be considered when calculating how much stress is in the structure or how much the chassis deflects. The user can determine the initial value for chassis mass when defining inputs to the model. The variable defining the initial chassis mass is called chassisMassFeedback.

To ensure that the model runs without error, a saturation block is placed after the state space feedback transfer function. The saturation block contains a lower limit of 0.001 [kg] and an upper bound of 500 [kg]. These bounds should never be hit by a feasible robot. They are in place to prevent the vehicle's mass from potentially increasing without bounds, which would cause the model to crash. Like the saturation block used to ensure that maximum velocity does not increase without bounds, the saturation block used to prevent vehicle mass from increasing without bounds is never utilized during the trade space exploration presented in this thesis. The saturation block remains in the model to prevent an error that could possibly result from the user selecting input parameters outside of the minimum and maximum bounds sampled by ATSV in this trade study.

The maximum deflection is at the center of the chassis as shown in Figure 22. The equation for the maximum deflection is shown in Equation 47 [27]. Solving this equation for the deflection limited chassis thickness results in Equation 48.

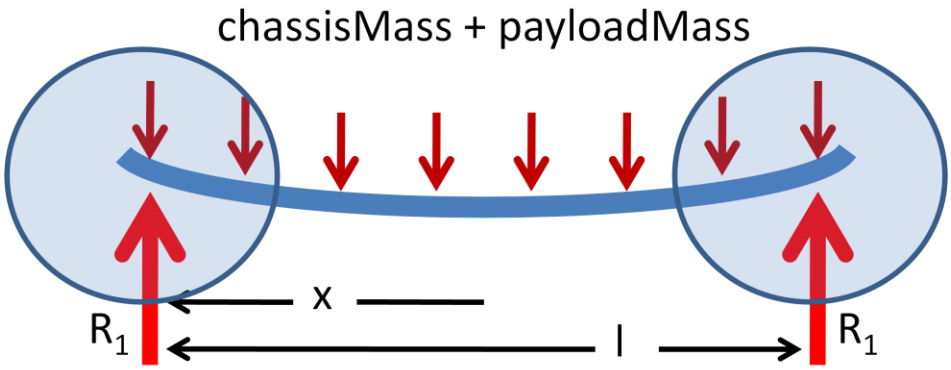

Figure 22: Chassis Maximum Deflection and Stress Schematic

$$maximumDeflection = \frac{5}{384} * \frac{W * chassisLength^3}{E * I} \tag{47}$$

where:

$$I = \frac{L * chassisThickness^3}{12}$$

and

$$W = (chassisMass + payloadMass) * g$$

$$chassisHeight = \sqrt[3]{\frac{60}{384} \frac{(chassisMass + payloadMass) * g * chassisLength^3}{E * chassisWidth * maxDeflection}} \tag{48}$$

The maximum stress also occurs at the center of the chassis. The maximum chassis stress is calculated using Equation 49. Solving Equation 49 for chassis thickness results in the stress limited chassis thickness shown in Equation 50.

$$\sigma_{max} = \frac{M * y}{I} \tag{49}$$

where:

$$I = \frac{chassisLength * chassisThickness^3}{12}$$

and

$$M = R * \left(\frac{chassisLength}{2} - \frac{wheelDiameter}{2}\right)$$

where:

$$R = \frac{(chassisMass + payloadMass) * g}{2}$$

$$chassisThickness = \sqrt{\frac{3 * (chassisMass + payloadMass) * g * \left(\frac{chassisLength}{2} - \frac{wheelDiameter}{2}\right)}{chassisWidth * \sigma_{max}}} \tag{50}$$

Once the deflection-limited and the stress-limited chassis thicknesses have been calculated, the larger (or thicker) value of the two is chosen. Chassis mass is then determined by multiplying the chassis material density by the volume of the chassis structure. Volume is simply the product of the chassis length, width, and thickness. This calculation for chassis mass is then used to recalculate chassis thickness.

3.7 Wheels

3.7.1 Subsystem Overview

Because a robot is specified by the user as either a wheeled vehicle or a tracked vehicle these two subsystems are evaluated side-by-side within the model as shown in Figure 23. Once outputs to both the Wheels and Tracks subsystems have been calculated, they all enter a block that selects the wheel output parameters if the user specifies a wheeled vehicle or the track output parameters if the user specifies a tracked vehicle.

These subsystems are discussed independently from one another; however, there are eight outputs from this Wheels/Tracks subsystem combination. These outputs include the vehicle wheelbase [m], the mass of the wheels or tracks [kg], the mass of the tracked vehicle drive sprockets [kg], width of the wheels or tracks [m], additional width of the vehicle necessary if the wheeled vehicle is not skid-steer capable [m], the thickness of the track [m]. Additionally two flags ensure that the design is feasible. The first flag is to let the user know that the width of the wheels had to be increased in order to satisfy the user-defined ground pressure requirement. The second flag lets the user know that the user-defined wheel diameter is greater than the chassis length. If the wheel diameter is greater than the chassis length, then the wheels would rub on one another. In both cases, a flag equal to one indicates the robot is infeasible.

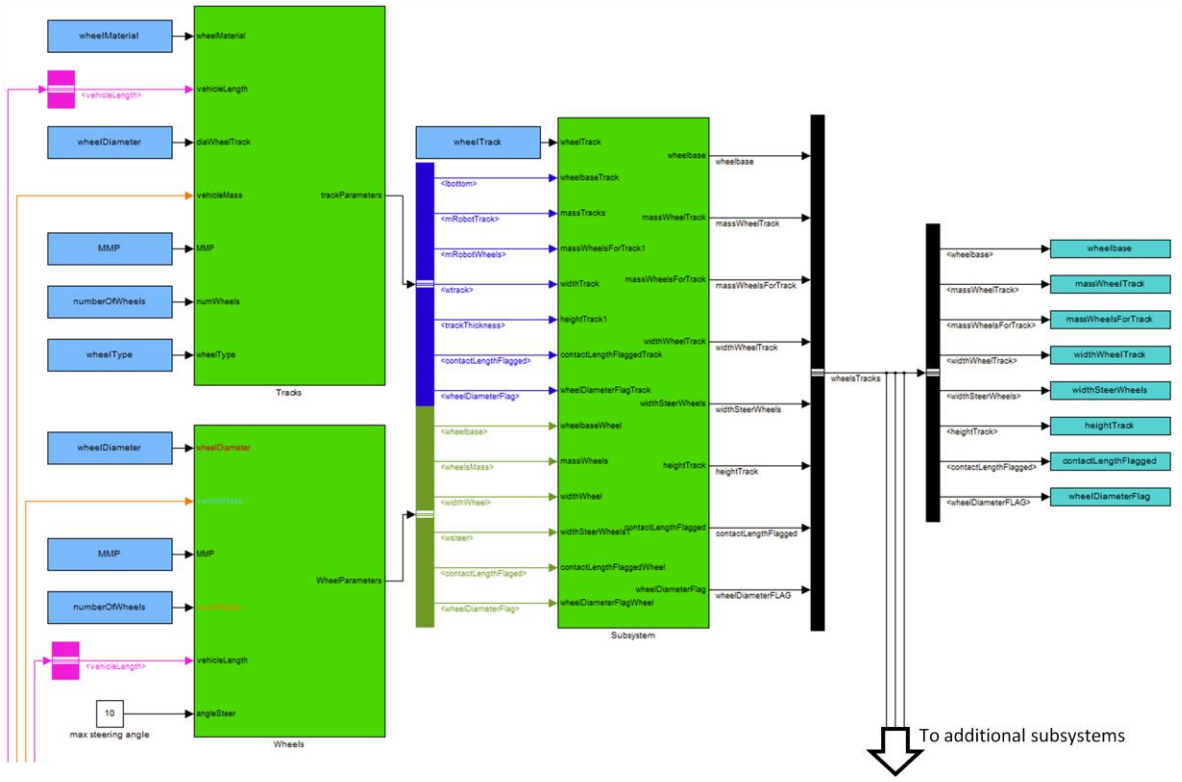

Figure 23: Wheels/Tracks Subsystem Combination

As can be seen in Figure 23, the outputs from the combined Wheels and Tracks subsystem are used as inputs in three other subsystems. These three subsystems are Total Vehicle Dimensions, Total Vehicle Mass, and Functional Capabilities.

As shown in Figure 24, the Wheels subsystem depends on two model-dependent inputs. The first is vehicle mass [kg], which is calculated using the feedback loop described in Section 2.5. The second model-dependent input is vehicle length [m]. The subsystem also depends on three user-defined inputs. These inputs are wheel diameter [m], ground pressure (shown in the model as MMP: Max-Mean-Pressure) [N/m^3], and the number of wheels on the vehicle.

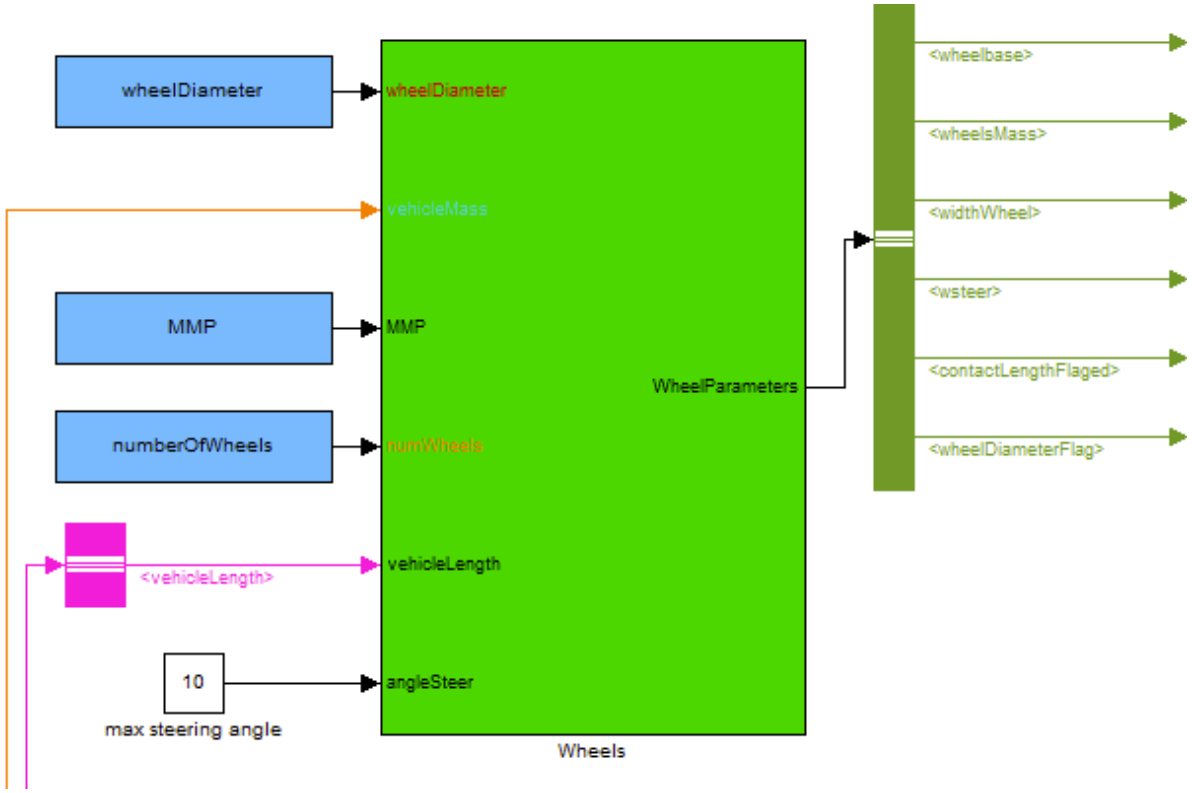

Figure 24: Simulink® Wheels Subsystem

3.7.2 Model Derivation

The Wheels subsystem first calculates the width of the wheels based on a correlation based on wheel diameter found in production wheels [26]. This correlation is found in Equation 51.

$$widthWheels = 0.425 * wheelDiameter \tag{51}$$

This calculated wheel width is then input into an iterative logic loop. The first block within the loop determines if the contact length flag is triggered. The calculated value from Equation 51 should be used on the initial pass; therefore, there is initially no flag, i.e.,

contactLengthFlagged=0. If the flag is triggered, i.e., if contactLegnthFlagged=1, then the loop increments the width of the wheels by 0.1 cm.

The next block determines the contact patch length based on the user-defined ground pressure and number of wheels as well as model-defined inputs of vehicle mass and wheel width. Contact length is derived as shown in Equation 52.

$$contactLength = \frac{vehicleMass * g}{widthWheels * numberOfWheels * groundPressure} \quad (52)$$

The next block triggers a flag (contactLengthFlagged=1) if the contact patch length is greater than 40% of the wheel diameter. The value of 40% was arbitrarily chosen, on the high side, to allow larger mud-bogging tires to be populated within the tradespace. If the block triggers a flag then the wheel width is increased by 0.1cm, and this process is repeated until the flagging condition is satisfied.

If a flag is not triggered, then the loop finishes, and the mass of the wheels is calculated via Equation 53. The value for density is hard-coded into the subsystem as 313.67 kg/m^3 as determined as an approximated average density of rubber tire treads with an aluminum rim [26].

$$wheelsMass = numberOfWheels * \left[\pi * \left(\frac{wheelDiameter}{2}\right)^2 * widthWheel\right] * density \quad (53)$$

Equation 54 is used to calculate the additional width needed for turning the wheels when steering. In this equation, angleSteer is hard-coded into the subsystem as 10 [degrees].

$$wsteer = \frac{wheelDiameter}{2} * \text{sind}(angleSteer) + \frac{widthWheel}{2} * cosd(angleSteer) - \frac{widthWheel}{2} \quad (54)$$

The wheel diameter flag is triggered when the vehicle length is less than two times the wheel diameter. Thus an infeasible vehicle would have wheelDiameterFlag=1. Again, this flag tells the user that the wheels are so big that they overlap and are not able to turn. Finally, the vehicle's wheelbase is calculated as the vehicle length minus the wheel diameter.

3.8 Tracks

3.8.1 Subsystem Overview

The Tracks subsystem is evaluated in parallel to the Wheels subsystem. The subsystem is evaluated similarly to the Wheels subsystem. As shown in Figure 25, there are two model determined inputs. Like the Wheels subsystem, these inputs are vehicle length [m] and vehicle mass [kg]. Additionally, there are five user-defined inputs to the system. The first is the wheel material selection, which has a dimensionless input ranging from 1 – 4: 1 is aluminum (2700 [kg/m^3]), 2 is carbon fiber (1760 [kg/m^3]), 3 is steel (7850 [kg/m^3]), and 4 is the Talon's composite material (2000 [kg/m^3]). The other user-defined inputs are wheel diameter [m], ground pressure [N/m^3], the number of wheels, and a parameter to select wheel type. The input range for wheel type ranges from 1 – 3: 1 is a 0.0063 [m] (1/4 inch) thick thin disk, 2 is an I-beam with disk thickness of 0.0063 [m] with a cylindrical thickness of 0.0021 [m] (1/8 inch), and 3 is a solid cylinder the entire width of the track.

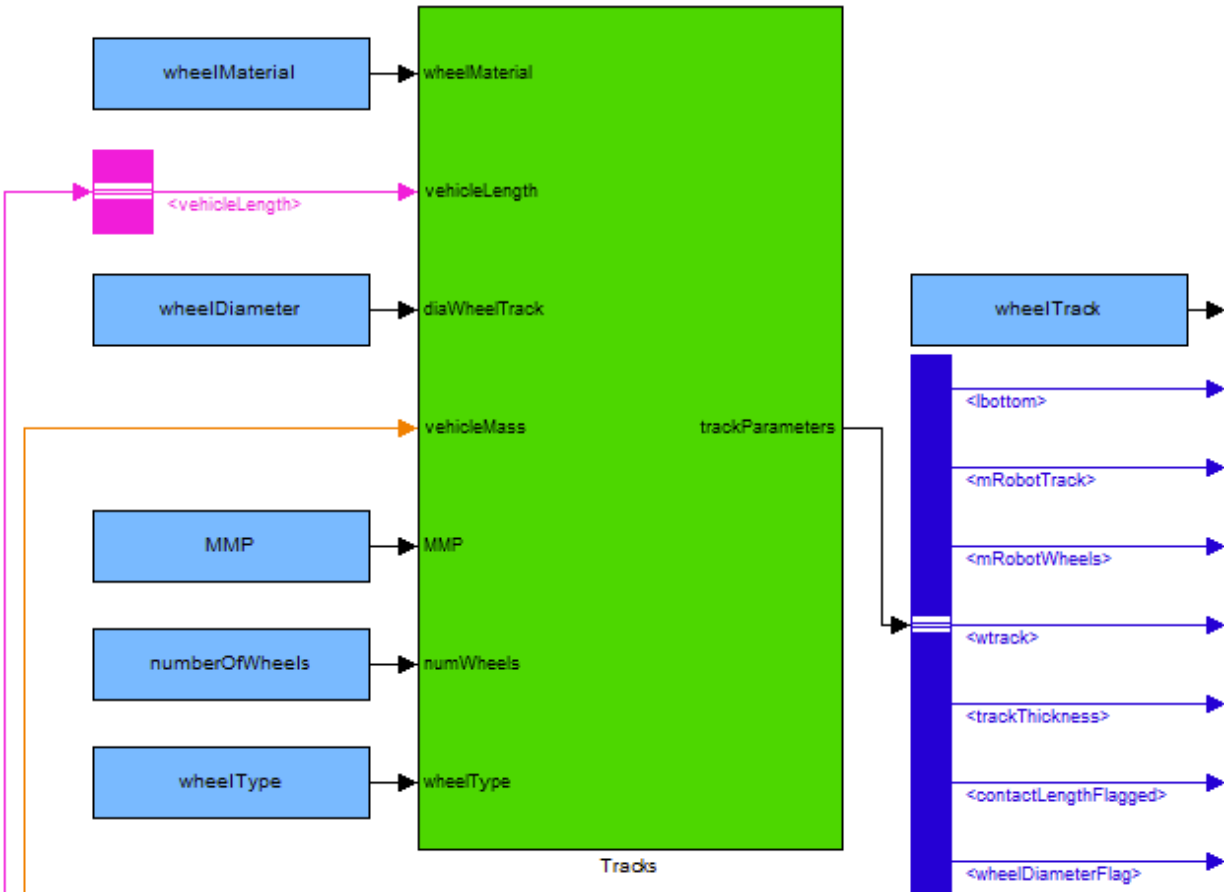

Figure 25: Simulink® Tracks Subsystem

3.8.2 Model Derivation

Like the Wheels subsystem, the Tracks subsystem determines if the wheels rub by outputting wheelDiameterFlag=1 if the vehicle length is less than two times wheel diameter. Wheelbase is calculated as the vehicle length minus wheel diameter. This value is then used to determine the width of the track, as shown in Equation 55.

$$widthTrack = \frac{vehicleMass * g}{2 * wheelbase * GroundPressure} \tag{55}$$

Track thickness is calculated as a function of vehicle mass. This function was created by scaling thickness based on the Talon. The function is shown in Equation 56.

$$trackThickness = \frac{(trackThicknessTalon = 0.004825m)}{(massTalon = 54kg)} * vehicleMass \tag{56}$$

Track width and thickness are then used to determine the volume of the drive sprocket and tire. The diameter of the rim is first calculated as shown in Equation 57. The volume of the rim is then calculated as shown in Equation 58. Representations of the three different wheel types are shown in Figure 26. Finally, the volume of the tire is calculated as per Equation 59.

$$diaRim = wheelDiameter - 2 * trackThickness \tag{57}$$

$$if \begin{bmatrix} wheelType = 1 \\ wheelType = 2 \\ wheelType = 3 \end{bmatrix} then\ rimVolume$$

$$= \begin{bmatrix} \frac{\pi}{4} * diaRim^2 * 0.0063 \\ \frac{\pi}{4} * [diaRim^2 - (diaRim - 2 * 0.0021)^2] * widthTrack + \frac{\pi}{4} * diaRim^2 * 0.0063 \\ \frac{\pi}{4} * diaRim^2 * widthTrack \end{bmatrix} \tag{58}$$

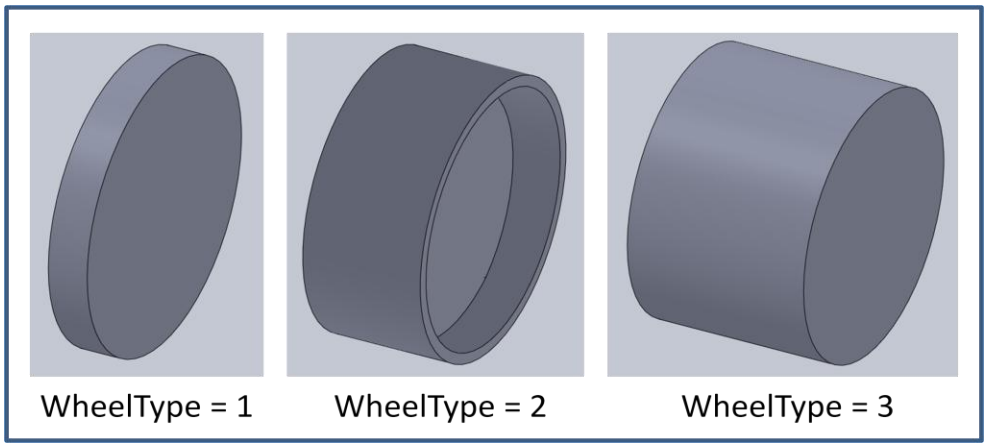

Figure 26: Three Wheel Type Graphical Representations

$$if \begin{bmatrix} wheelType = 1 \\ wheelType = 2 \\ wheelType = 3 \end{bmatrix} then\ tireVolume$$

$$= \begin{bmatrix} 0 \\ \frac{\pi}{4} * [wheelDiameter^2 - (diaRim - 2 * trackThickness)^2] * widthTrack \\ \frac{\pi}{4} * [wheelDiameter^2 - (diaRim - 2 * trackThickness)^2] * widthTrack \end{bmatrix} \quad (59)$$

The mass of all of the sprockets (generically called wheels when combined with the wheels subsystem) on the tracked vehicle are then calculated using Equation 60. The density of the wheels in this equation is user-definable based on the wheelMaterial input. Currently, the density of the tires is hard-coded into the subsystem as that of aluminum (2700 [kg/m^3]).

$$massWheels = numberWheels * (\rho_{wheel} * rimVolume + \rho_{tire} * tireVolume) \quad (60)$$

The final output to be calculated in the subsystem is the mass of the track. As shown in Equation 61, the mass of both of the tracks depend on the track geometry as well as the material

density, which is hard-coded into the model as 404.18 [kg/m^3]. This density was approximated from the Talon's track.

$$massTracks = 2 * \rho_{track} * volumeTrack \tag{61}$$

where:

$$volumeTrack = lengthTrack * widthTrack * trackThickness$$

where:

$$lengthTrack = 2 * wheelbaseLength + \pi * wheelDiameter$$

3.9 Power Requirements

3.9.1 Subsystem Overview

The Power Requirements subsystem follows the Wheels and Tracks subsystems. As seen in Figure 27, this subsystem depends on seven model-defined input variables. The subsystem depends on chassis height and chassis width, both of units [m], from the Chassis Dimensions subsystem. It depends on vehicle mass [kg]. The total battery power is the only model-defined input from the Batteries subsystem, and motor controller efficiency is the only input from the Motor Controller subsystem. Finally, the gearbox efficiency and motor efficiency are calculated in the Drive Motor subsystem. The user-defined inputs include selecting a wheeled or tracked vehicle configuration, ground clearance [m], and the average auxiliary power draw [W].

The Power Requirements subsystem has a single output, namely, maximum vehicle velocity [m/s].

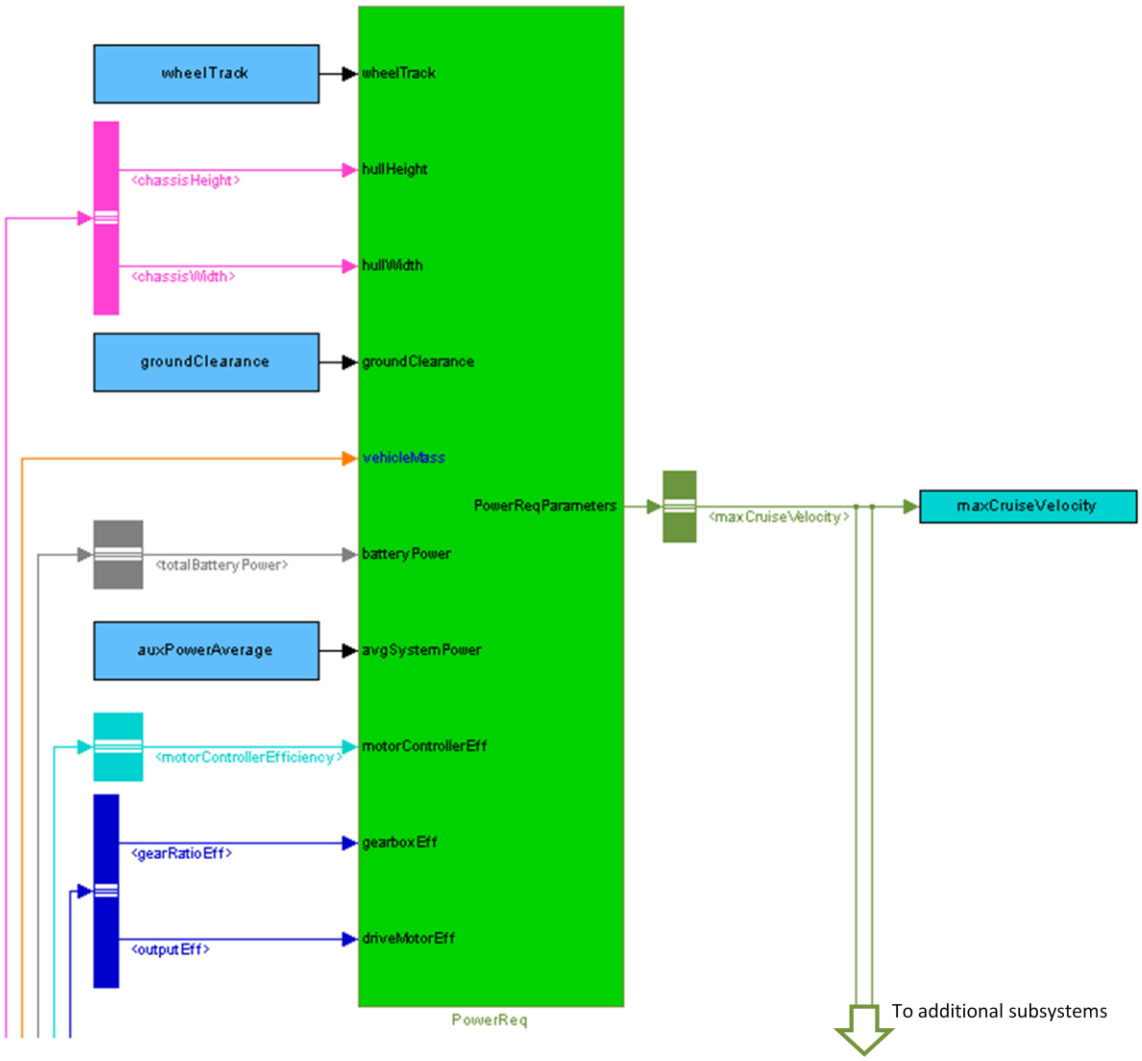

Figure 27: Simulink® Power Requirements Subsystem

The only output from this subsystem is used as an input to only one other subsystem, the Endurance subsystem. Additionally, as mentioned in Section 2.5, the output of this subsystem is fed back through the transfer function as shown in Figure 6.

3.9.2 Model Derivation

The Power Requirements subsystem calculates all of the resistive forces on the vehicle and then determines how fast the vehicle can drive based on how much power is available from the batteries.

Like the Chassis Structure subsystem, the Power Requirements subsystem also contains a feedback loop within the subsystem. Therefore, the first step in this subsystem is to guess an initial maximum vehicle velocity. The user can define this initial value, maxCruiseVelocity, at the same time as all of the input parameters are defined; however, the model converges regardless of what this value is.

The first calculations to be made are those to determine the forces resisting vehicle motion. If a wheeled vehicle configuration is selected, then the forces that resist vehicle motion are rolling resistance and wind resistance. The power loss associated with rolling resistance is a function exclusively of vehicle weight. The rolling resistance equation is shown in Equation 62, where Crr is the dimensionless coefficient of rolling resistance on asphalt equal to 0.0133 [28].

$$rollingResistance = Crr * (vehicleMass * g) \tag{62}$$

Wind resistance is nearly negligible; however, it does have a slight impact on resistive power loss, especially at higher velocities. The equation to calculate the forces due to wind resistance is shown in Equation 63. In this equation, the dimensionless coefficient of wind resistance, cW is equal to 0.9, and the density of air is 1.2 [kg/m^3]. Additionally, the vehicle's frontal area [m^2] is approximated as the quantity of the hull thickness plus ground clearance multiplied by the chassis width.

$$windResistance = \frac{cW * \rho_{air} * maxCruisingVelocity^2 * frontalArea}{2} \quad (63)$$

If a tracked vehicle configuration is selected, then the resistive forces include an experimentally determined power loss and terrain resistance loss. The equation for the experimentally determined power loss was calculated by measuring the power consumption of the Tankbot driving on various surfaces while varying speed [26]. This curve fit is shown in Equation 64. Additionally, the power consumption due to plowing effects on asphalt is shown in Equation 65.

$$drivetrainPowerLoss = vehicleMass * (1.944 * maxCruisingVelocity + 0.054) \quad (64)$$

$$terrainResistance = 0.0128 * vehicleMass * maxCruisingVelocity^{2.06} \quad (65)$$

These resistive forces are then summed together and increased by dividing by the product of the motor controller, gearbox, and drive motor efficiencies. The average auxiliary power draw is added to this value to obtain the total power required to drive the vehicle at the designated speed. Finally the maximum vehicle velocity is calculated using Equation 66.

$$maxCruisingVelocity = \frac{totalBatteryPower}{tractionEffort} \quad (66)$$

As previously mentioned, the resistive forces due to wind, drivetrain power loss, and terrain loss depend on the maximum vehicle velocity. Therefore, once the maximum vehicle

velocity has been calculated, it is passed through a feedback loop to ensure convergence of the parameter. Like the feedback loop for maximum vehicle velocity in the high level view of the model, the feedback loop within this subsystem contains a saturation block with an upper limit of 1000 [m/s] and a lower limit of 0.01 [m/s].

3.10 Endurance

3.10.1 Subsystem Overview

As shown in Figure 28, the Endurance subsystem depends on eight model-defined inputs. Because this subsystem calculates how far the vehicle can drive based on the batteries, the subsystem is highly dependent on outputs from the Battery subsystem. The three model-defined inputs from the Battery subsystem are the dimensionless Peukert Number, total battery power [W], and battery capacity with depth of discharge factored [Amp-hrs]. The Endurance subsystem is dependent on both of the iteratively solved for parameters in the model: vehicle mass [kg] and maximum vehicle velocity [m/s]. The Endurance subsystem depends on the same three efficiency losses as the Power Requirements subsystem: motor controller, gearbox, and drive motor efficiency. The Endurance subsystem also depends on two user-defined input parameters: the average auxiliary power draw [W] and bus voltage [V].

The subsystem calculates two parameters. The first is battery time [hr], which is how long the vehicle can drive at the maximum vehicle velocity. The second is battery distance [km], which is defined as the distance that the vehicle can drive at the maximum vehicle velocity. Additionally, the subsystem outputs a flag (batteryVelocityFlag=1) if the vehicle does not have enough battery power to drive at this velocity. Thus, if the value for the maximum cruising distance is zero, the battery velocity flag is set equal to one.

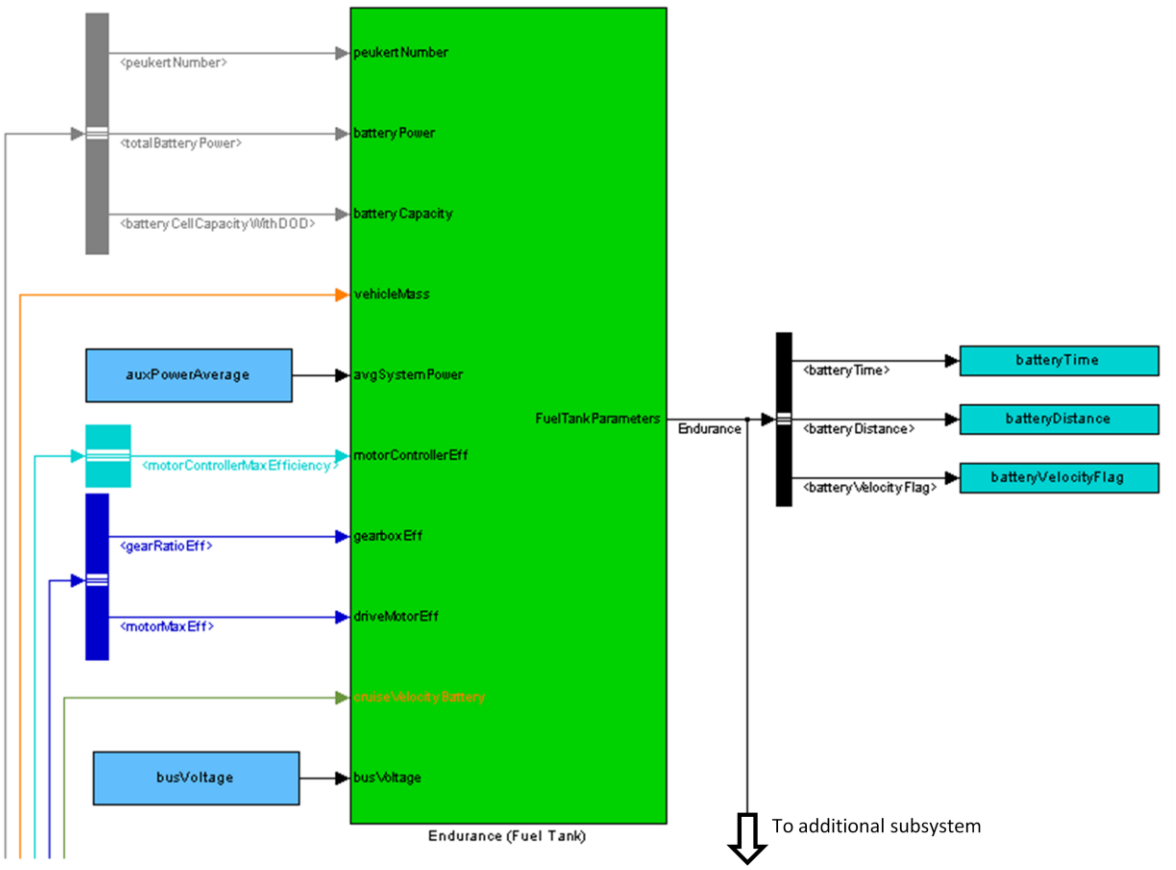

Figure 28: Simulink® Endurance Subsystem

The outputs from this subsystem are used as inputs to the Effectiveness subsystem.

3.10.2 Model Derivation

In order to obtain a better model prediction to actual measurement correlation, a different set of equations was used to determine the resistive force acting on the vehicle than were used in the Power Requirements subsystem. The two resistive losses are shown in Equation 67 and Equation 68. The total power required to drive the vehicle with no inefficiencies at the maximum vehicle velocity, in units of [W], is the sum of these two values [26].

$$loss1 = (1.1848 * maxCruiseVelocity + 0.1531) * vehicleMass \quad (67)$$

$$loss2 = 0.944 * maxCruiseVelocity^{2.06} * \frac{vehicleMass}{74.4} \quad (68)$$

Just as in the Power Requirements subsystem, the total power required to drive the robot is increased by dividing the power required to overcome the resistive losses by the product of the motor controller, gearbox, and drive motor efficiencies. This power is then added to the average auxiliary power draw to obtain the total power required from the batteries in order to drive the robot and power all system devices.

The total power required from the batteries is then used by a Modified Peukert Number function, shown in Equation 69, to determine how long, in units of [hrs], the batteries can supply the required power [26]. In this equation the rated time of the battery is assumed to be 20 [hrs]. Discharge current is calculated as the power supplied by the batteries divided by the bus voltage. Battery capacity [Amp-hrs] and Peukert Number are calculated in the Batteries subsystem. Finally, the distance [km] is calculated as the battery time multiplied by the maximum cruise velocity.

$$batteryTime = batteryRatedTime\left(\frac{batteryCapacityWithDOD}{dischargeCurrent * batteryRatedTime}\right)^{peukertNumber} \quad (69)$$

3.11 Total Vehicle Dimensions

3.11.1 Subsystem Overview

The Total Vehicle Dimensions subsystem is a very simple subsystem that sums the necessary vehicle parameters to obtain the vehicle width [m] and height [m]. The subsystem sums the total torque [N-m] available from the drive motors and gearboxes. The subsystem also calculates centers of gravity [m] in three dimensions. CGy is defined from either side of the vehicle to the center of gravity. CGySAE is defined from the geometric center to the center of gravity; thus, CGySAE is always equal to zero. GCx is the distance from the rear of the vehicle to the center of gravity while CGxSAE is defined from the rear axle to the center of gravity. CGz and CGzSAE are both defined from the ground to the center of gravity.

In order to calculate these parameters, as shown in Figure 29, the subsystem depends on the outputs from five other subsystems. Total Vehicle Dimensions depends on the chassis width [m] and vehicle length [m] from the Chassis Dimensions subsystem. It depends on the width [m] and mass [kg] of the wheels or tracks from the Wheels or Tracks subsystem. It uses the chassis structure thickness [m] and chassis mass [kg] from the Chassis Structure subsystem. The number of drive motors, drive motor mass [kg], and maximum drive motor/gearbox output torque [N-m] are outputs from the Drive Motor subsystem. Total vehicle height depends on the radii of each arm segment [m], which are sized in the Manipulator subsystem. Finally, the total manipulator mass [kg] is required to calculate the center of gravity in the z-direction.

The subsystem also relies on six user-defined input parameters including a binary wheel or track configuration, ground clearance [m], payload height [m], internal payload mass [kg], wheel diameter [m], and the number of manipulator segments.

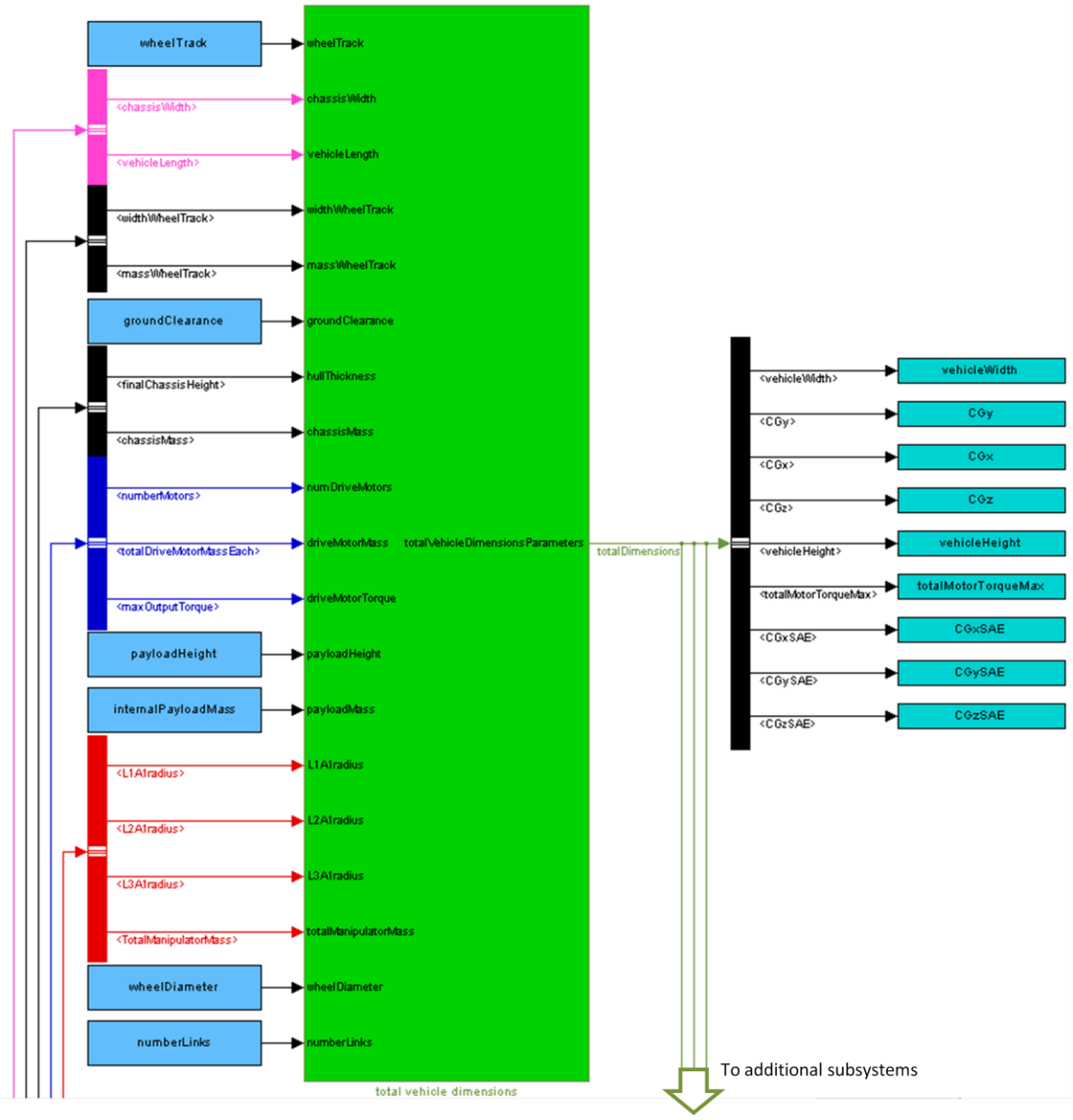

Figure 29: Simulink® Total Vehicle Dimensions Subsystem

The outputs of the Total Vehicle Dimensions subsystem are used in three of the final four remaining subsystems including Functional Capabilities, Manipulator Capabilities, and Effectiveness.

3.11.2 Model Derivation

The total vehicle width is calculated differently for wheeled and tracked vehicles. On a tracked vehicle, half of the width of each track is assumed to cover the chassis, whereas this is not possible for a tracked vehicle. Thus the total vehicle width for a wheeled vehicle is the chassis width [m] plus two times the width of the wheels [m]. The total vehicle width for a tracked vehicle is the chassis width [m] plus two times the width of the tracks [m]. The value for CGy [m], the distance from one side of the vehicle to the center of gravity, is then determined as half the value for the total vehicle width. Because the center of gravity in the width direction of the vehicle, y, is assumed to be at the geometric center, CGySAE [m], the distance from the geometric center of the vehicle to the center of gravity, is assumed to be zero.

The value for CGx [m], the distance from the rear of the vehicle to the center of gravity is assumed to be half of the vehicle length. CGxSAE [m] is calculated as CGx subtracted by the radius of a wheel, to give the distance from the rear axle to the center of gravity.

CGz [m], the distance from the ground to the center of gravity, is calculated by dividing the moment about the ground due to each of the components of the vehicle by the masses of each component. The relative heights of each component with respect to one another are also used to determine the total height of the vehicle [m]. CGzSAE [m] is equal to CGz, as it is also defined as the distance from the ground to the center of gravity.

Total drive motor torque is simply the number of drive motors multiplied by the torque of each motor.

3.12 Total Vehicle Mass

The Total Vehicle Mass subsystem is more of a state-space summation block than it is an analytical subsystem like the previously discussed subsystems. Nonetheless, this subsystem sums all of the component masses together to calculate a parameter called total vehicle mass. Because many subsystems depend on vehicle mass as an input, the final convergent value for the total vehicle mass must be solved for using a feedback loop described in Section 2.5.

As shown in Figure 30, the masses that are summed together in this subsystem are the mass of the wheels or tracks from the Wheels or Tracks subsystem, the user-defined value for internal payload mass, the chassis mass from the Chassis Structure subsystem, the total battery mass from the Battery subsystem, the mass of the user-defined battery controls, the mass of the user-defined sensor package, the mass of the motor controllers from the Motor Controller subsystem, the mass of the drive motors from the Drive Motor subsystem, and finally the total mass of the manipulator from the Manipulator subsystem. All of these masses are in [kg].

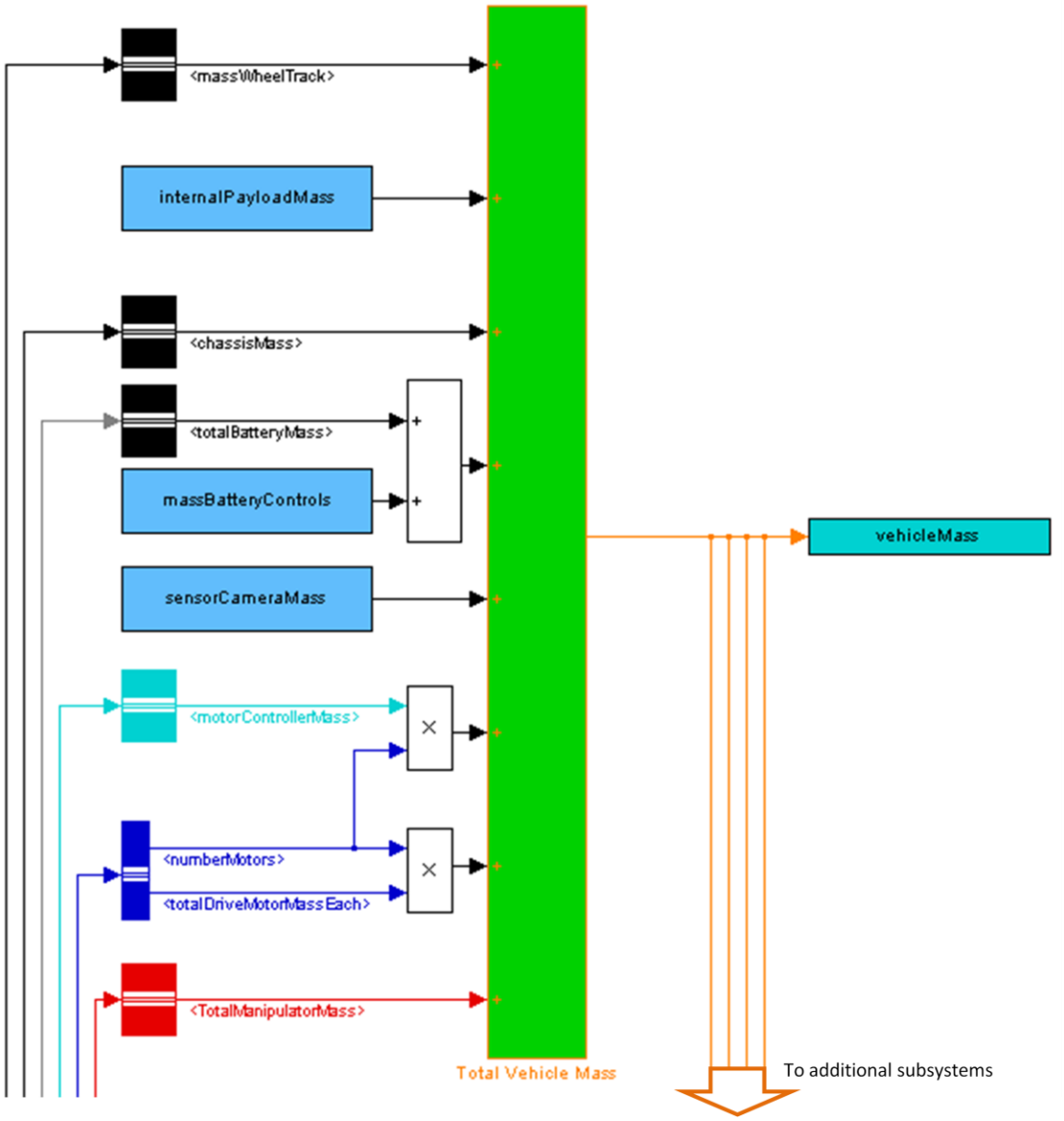

Figure 30: Simulink® Total Vehicle Mass Subsystem

As with maximum vehicle velocity in Section 2.5, convergence of the total vehicle mass was confirmed by proving the final value theorem, i.e., by defining the initial conditions for the three interdependent variables as the final values to which they are expected to converge.

The total vehicle mass is used as an interdependent input to four subsystems: Endurance, Power Requirements, Wheels, and Tracks. Total vehicle mass is also an input to all three

remaining subsystems in the model: Functional Capabilities, Manipulator Capabilities, and Effectiveness.

3.13 Functional Capabilities

3.13.1 Subsystem Overview

At this point in the model, the robot's specifications have been completely defined. The final three subsystems, including Functional Capabilities, are merely subsystems to determine functionality of the robot design.

As shown in Figure 31, the Functional Capabilities subsystem depends on the robot's center of gravity in all three directions (CGxSAE, CGzSAE, and CGySAE) [m] and vehicle width [m] from the Total Vehicle Dimensions subsystem. It depends on the quantity and torque [N-m] of the drive motors from the Drive Motors subsystem. Total vehicle mass [kg] in an input to the subsystem. The vehicle's length [m] is calculated in the Chassis Dimensions subsystem. The wheelbase [m] and width of the wheels or tracks [m] are calculated in the Wheels or Tracks subsystem.

The Functional Capabilities subsystem depends on three user-defined input parameters. Two of them have been previously described: wheel diameter [m] and selection of a wheeled or tracked configuration. Additionally, the friction-based failure modes within the subsystem depend on a coefficient of static friction.

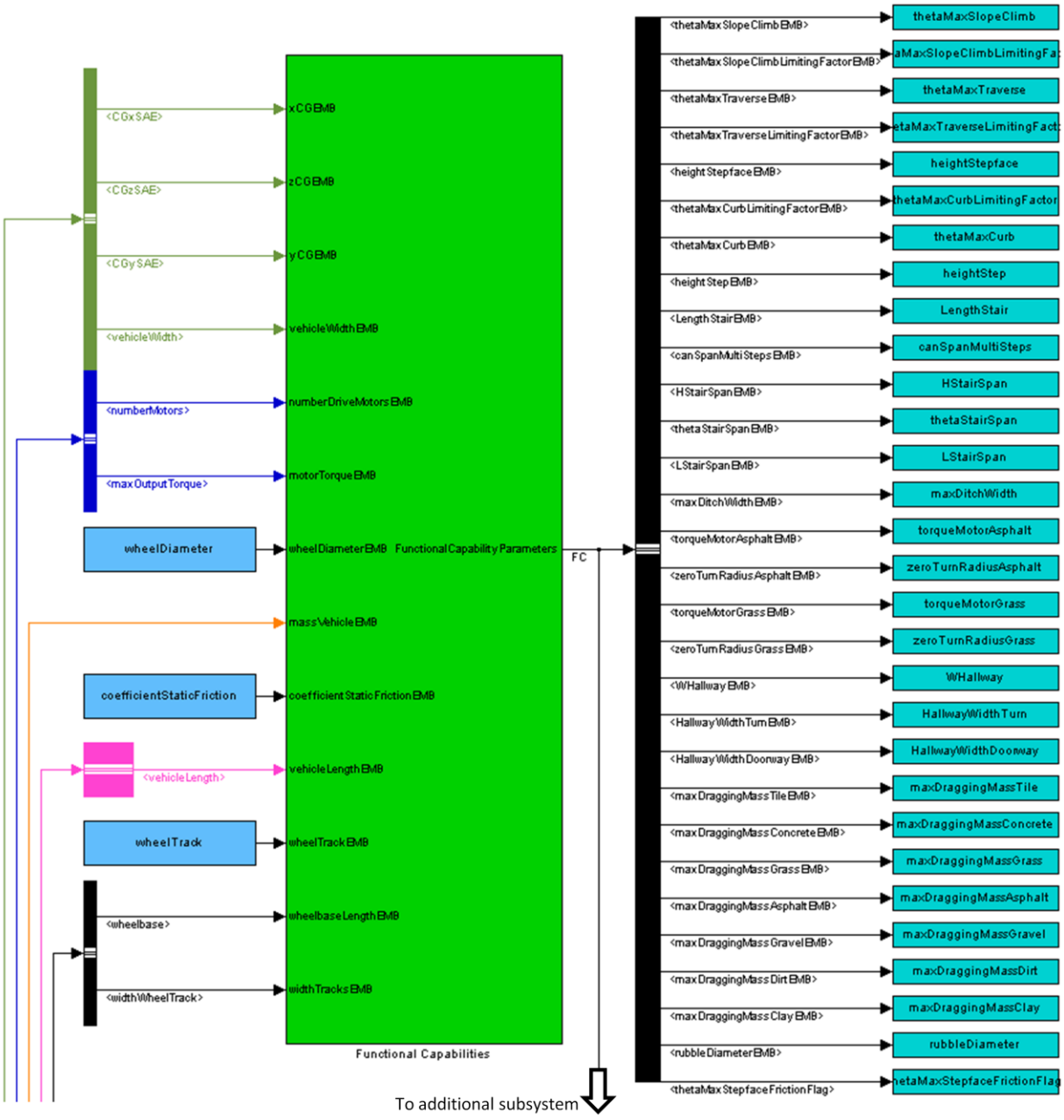

Figure 31: Simulink® Functional Capabilities Subsystem

3.13.2 Subsystem Derivation

Because the Functional Capabilities subsystem is described in significant detail in another work [26], only an outline of this subsystem is provided.

This subsystem outputs the maximum slope that the robot can climb based on three possible failure modes: (1) tipping, (2) torque, and (3) friction. The subsystem also indicates in which of these three conditions the vehicle fails. The maximum slope is determined by calculating the maximum climbable angle based on tipping, torque, and friction independent of one another and then selecting the minimum of the three angles. The maximum traversable slope is calculated in a similar manner; however, it has only two possible failure modes: (1) tipping and (2) friction. Like slope climbing, the failure mode is output from the subsystem.

Next, curb climbing is evaluated. A curb is defined as a single step. Curb climbing capability has six failure modes: three for climbing the face of the step and three for climbing up over the step. Climbing the face of the step is limited by (1) tipping, (2) torque, and (3) friction. Climbing up over the step is also limited by (4) tipping, (5) torque, and (6) friction. The maximum height of a curb [m] that can be ascended is determined using a trigonometric function. The maximum staircase climbing slope [deg] is determined in a similar fashion from which the maximum rise [m] and run [m] of the staircase stairs are determined.

The maximum width of a ditch that the vehicle can traverse [m] is calculated differently for a wheeled and a tracked vehicle. In either case, the maximum traversable distance is calculated using simple geometric parameters of the vehicle.

A static force analysis is done on the vehicle to determine if the drive motors can provide enough torque to perform a zero degree radius turn. This analysis is done on two surfaces: asphalt and grass. If the vehicle can perform a zero degree radius turn, a zero is output from the model; if it cannot, a one is output.

Finally, the minimum hallway width [m] in which the vehicle can maneuver is calculated as a function of geometric properties and an experimental deviation value.

3.14 Manipulator Capabilities

3.14.1 Subsystem Overview

The Manipulator Capabilities subsystem is essentially an extension of the Functional Capabilities subsystem but focuses on the capabilities of the manipulator.

As would be expected, the subsystem depends on several parameters calculated within the Manipulator subsystem. As shown in Figure 32, these parameters include the torque [N-m] and mass [kg] of each motor and gearbox combination within the manipulator. Also input from the Manipulator subsystem is the mass of each segment link. The Manipulator Capabilities subsystem also depends on many of the same user-defined input parameters as the Manipulator subsystem. These inputs include the gripper mass [kg], number of links, gripper length [m], length of each segment [m], and close lifting capacity (FLift) [kg].

The subsystem also depends on the final converged value of vehicle mass [kg], vehicle length [m] from the Chassis Dimensions subsystem, distance from the rear of the vehicle to the center of gravity (CGx) [m] from the Total Vehicle Dimensions subsystem, and drive motor diameter [m] from the Drive Motor subsystem. The subsystem also depends on the user-defined wheel diameter [m]. Finally, it depends on a parameter to adjust the location where the base of the manipulator connects to the vehicle's chassis: percentManipToCGx. This dimensionless parameter, varying from 0 to 1, allows the base motor of the manipulator to be mounted anywhere from just in front of the center of gravity to in line with the front axle.

The Manipulator Capabilities subsystem outputs five values. The first is a binary can/cannot (0/1) self-right if the vehicle happens to flip itself completely upside down. The second is the fully extended length of the manipulator. Third is the distance, from the front of the chassis, that the manipulator can extend without tipping over the robot solely due to the

weight of the arm. Tipping must be considered because it is possible to design a robot within the model that has a very light chassis and a very heavy manipulator. The fourth parameter is similar to the third; however, it is the distance in front of the chassis that the manipulator can extend without tipping the robot while lifting not only its own inherent mass but also the close lift capacity. Finally, it is possible to define a robot with one segment link that, because it can only touch the ground in its fully extended configuration, the robot is not able to lift the close lift capacity without tipping the robot. If a robot with this configuration is defined, then a flag is output (cantLiftFLiftFlag = 1).

Outputs from the Manipulator Capabilities subsystem are used only in the Effectiveness subsystem.

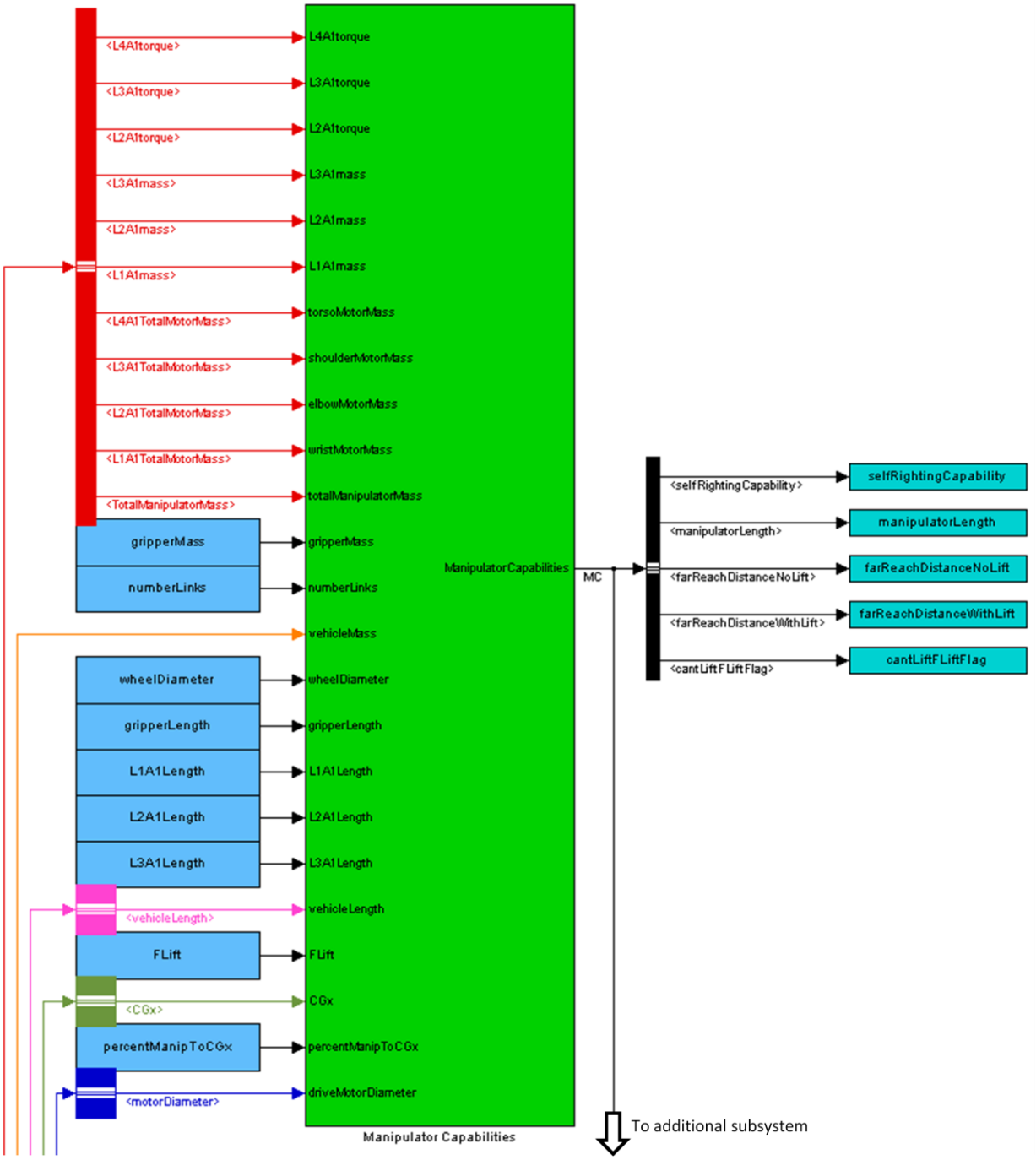

Figure 32: Simulink® Manipulator Capabilities Subsystem

3.14.2 Model Derivation

Self-Righting Capability

Self-righting capability is the ability of a robot to flip itself over if it ends up upside down. This condition, where the robot is oriented upside down, is often referred to as a "turtleback" as shown in Figure 33. Because of how the Manipulator subsystem is driven, the strongest motor in the manipulator is the motor fixed to the chassis. In addition, applying torque at this motor relieves the need to also lift the other links and motors of the manipulator. Therefore, torque is applied by the manipulator motor fixed to the chassis. This motor applies a torque that pushes the rest of the manipulator against the ground as shown in Figure 33.

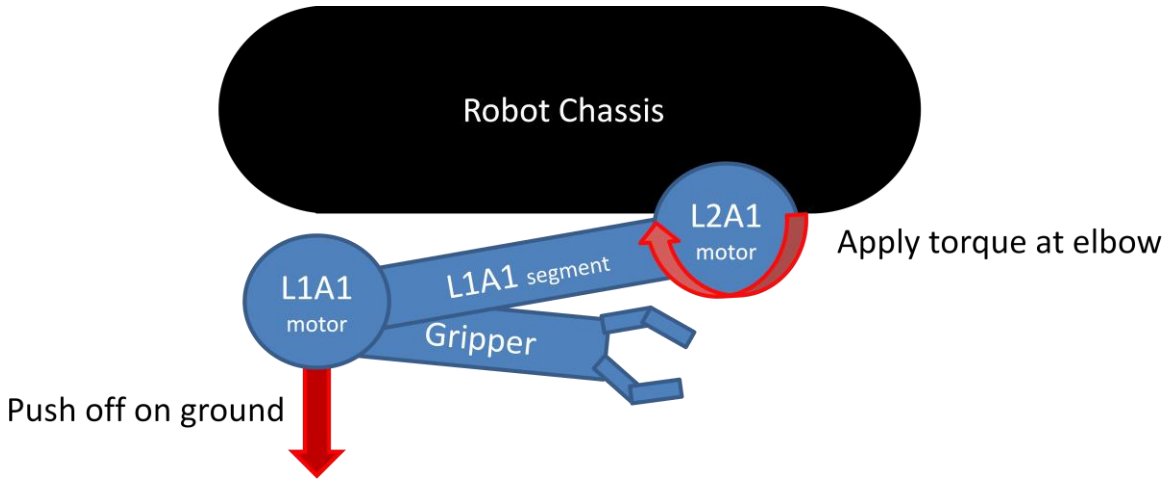

Figure 33: Turtleback Configuration

In order to self-right, this motor, motor L2A1 in Figure 33, must be able to provide enough torque to overcome the moment created by the mass of the chassis. This condition is described in Equation 70. In this equation, the parameter called percentManipToCGx was added after initial trade studies were performed yielding only light robots with very heavy manipulators

capable of self-righting themselves due to the increased torque required by the base manipulator motor when the manipulator is mounted far from the center of gravity.

$$baseManipulatorMotorTorque > chassisMass * g * xCGtoMotor \tag{70}$$

where:

$$xCGtoMotor = percentManipToCGx * \left(vehicleLength - CGx - \frac{wheelDiameter}{2}\right) + 0.001$$

Additionally, the segment link closest to the motor in question must be longer than the distance from the center of gravity to the mounting location of the base manipulator motor. This capability within the manipulator capabilities subsystem outputs a binary flag of 0 if the manipulator can self-right or 1 if it does not have a self-righting capability.

Far Reach Distance

There are three functional reach capabilities calculated in this subsystem. As mentioned in Section 3.14.1, the subsystem first calculates the maximum reach of the manipulator without considering tipping. This maximum reach is merely the total length of the manipulator as shown in Equation 71. Note that this equation is defined for a three segment manipulator. If fewer segments have been defined then their lengths are omitted from this calculation.

$$\begin{aligned} &totalManipulatorLength \\ &\quad = L3A1Length + L2A1Length + L1A1Length + gripperLength \end{aligned} \tag{71}$$

Far reach distance is next calculated as the maximum distance that a manipulator can reach without tipping the robot. This far reach distance is first calculated for the manipulator supporting only its own weight. It is also calculated for the manipulator lifting the close lift capacity at the to-be-determined far reach distance. The far reach distance for the manipulator supporting its own weight is simply output from the model for reference. The far reach distance while supporting the close lift capacity is used to determine effectiveness.

Far reach distance is calculated by balancing the moments about the point where the manipulator connects to the chassis. This connecting point labeled "CG Chassis to Manipulator" in Figure 34 varies from just in front of the chassis's center of gravity to the front of the chassis in line with the front wheels or sprockets. As mentioned in Section 3.14.2, this variance is controlled by a model input called percentManipToCGx.

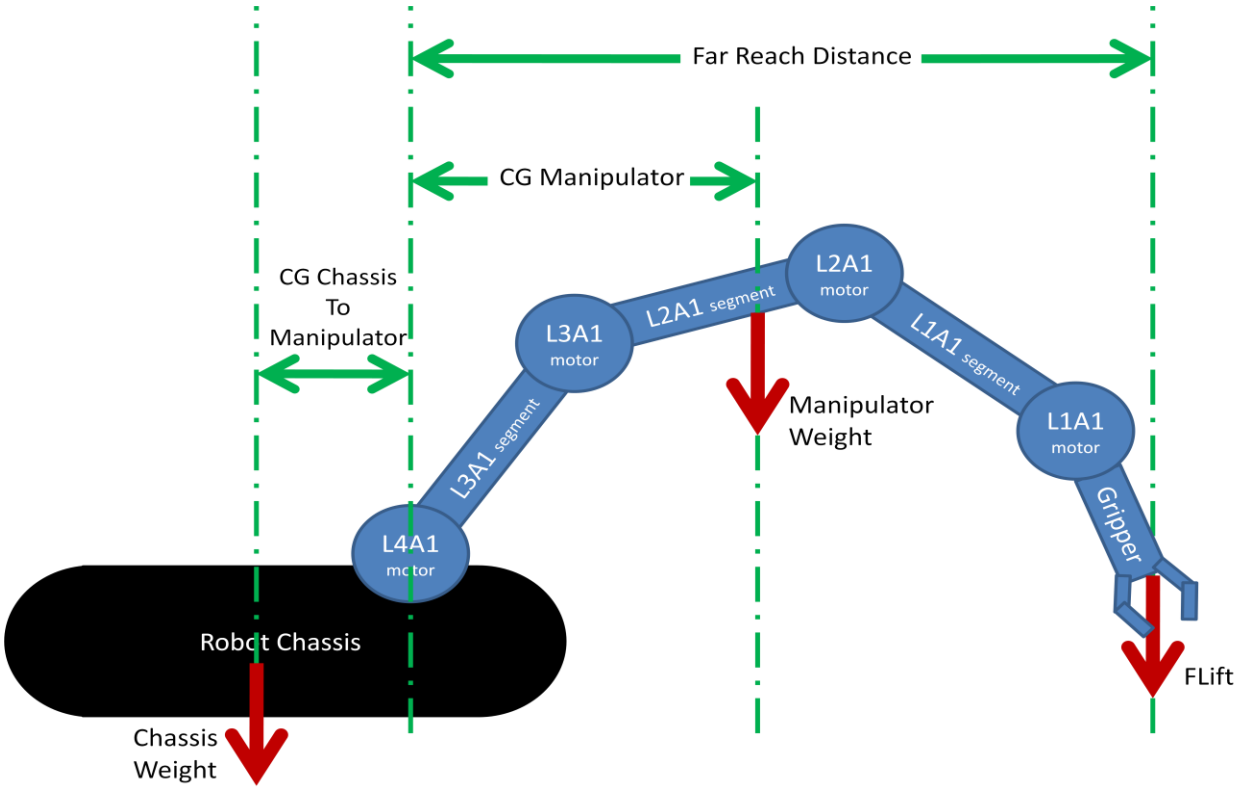

Figure 34: Far Reach Distance Configuration

There are two unknowns in Figure 34: (1) far reach distance and (2) the distance from the base of the manipulator to the manipulator center of gravity. Finding the equivalent center of gravity of the manipulator involves first determining the distance of each horizontally stretched manipulator component from its base. This equation for a three segment manipulator is given by Equation 72.

$$\begin{bmatrix} CG_x Motor_4 \\ CG_x Link_3 \\ CG_x Motor_3 \\ CG_x Link_2 \\ CG_x Motor_2 \\ CG_x Link_1 \\ CG_x Motor_1 \\ CG_x Gripper \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{L3A1Length}{2} \\ L3A1Length \\ L3A1Length + \frac{L2A1Length}{2} \\ L3A1Length + L2A1Length \\ L3A1Length + L2A1Length + \frac{L1A1Length}{2} \\ L3A1Length + L2A1Length + L1A1Length \\ L3A1Length + L2A1Length + L1A1Length + \frac{gripperLength}{2} \end{bmatrix} \quad (72)$$

The distances listed in Equation 72 are then used to sum the moment of each component at the location where the manipulator attaches to the chassis. The equation for this moment calculation is given for a three segment manipulator in Equation 73. Note that in this equation, gravity is not included because it cancels out in Equation 74 when dividing by the total manipulator weight.

$$\begin{aligned} \sum CG_{manipulator} \\ &= CG_x Motor_4 * L4A1motorMass + CG_x Link_3 * L3A1mass \\ &+ CG_x Motor_3 * L3A1motorMass + CG_x Link_2 * L2A1mass \\ &+ CG_x Motor_2 * L2A1motorMass + CG_x Link_1 * L1A1mass \\ &+ CG_x Motor_1 * L1A1motorMass + CG_x Gripper * gripperMass \end{aligned} \quad (73)$$

Using Equation 73, a scaling factor for the far reach distance is calculated. This scaling factor, shown in Equation 74, allows the distance from the base of the manipulator to the

manipulator's center of gravity to be approximated without knowledge of the relative angles that each segment has to one another.

$$FRD_{scalingFactor} = \frac{\sum CG_{manipulator}}{totalManipulatorMass * totalManipulatorLength} \tag{74}$$

Summation of the moments at the base of the manipulator from the entire robot results in Equation 75. The farthest distance that the manipulator can reach from the base of the manipulator is solved for using Equation 75. This distance is given in Equation 76. In the case of the manipulator supporting only its own weight, FLift [kg] is equal to zero.

$$\begin{aligned} \sum M_{manipulatorBase} &= 0 \\ &= FRD * FRD_{scalingFactor} * m_{manipulator} + FRD * F_{Lift} \\ &- CGchassisToManipulator * (m_{robot} - m_{manipulator}) \end{aligned} \tag{75}$$

$$FRD = \frac{xCGtoMotor * (m_{robot} - m_{manipulator})}{FRD_{scalingFactor} * m_{manipulator} + F_{Lift}} \tag{76}$$

where:

$$\begin{aligned} xCGtoMotor = &\left[vehicleLength - CGx - \frac{wheelDiameter}{2}\right] * (1 \\ &- percentManipToCGx) \end{aligned}$$

The distance that the arm must reach to first get past the front of the chassis is subtracted from the total far reach distance as given in Equation 77.

$$FRD_{actual} = FRD - (percentManipToCGx) * \left(vehicleLength - CGx - \frac{wheelDiameter}{2}\right) \quad (77)$$

A check is done as a final step to ensure that the manipulator is long enough to attain the far reach distance by selecting the minimum of the far reach distance, given by Equation 76, and the manipulator length, given by Equation 71. For this minimum function, the manipulator length first is subtracted by the length to get to the front of the chassis. Additionally, far reach distance is limited to a value of zero – such a value would indicate that if the robot extends its manipulator any more, then the robot would tip.

3.15 Effectiveness

The effectiveness subsystem is used to determine how well the simulated robot meets each of the pre-defined threshold and objective capability requirements. This subsystem is not discussed in detail because effectiveness metrics change drastically based on user needs.

The capability requirements vary based on the numerical value for size. Therefore, the size of the robot is first determined by determining if the simulated robot is a small, medium, or large robot. This is based purely on vehicle mass. From this point on, the subsystem simply evaluates how well each requirement is achieved, either as a binary yes or no or as a percent of the objective or threshold value. The subsystem first calculates how well the robot meets the threshold capabilities. Threshold values are the minimum values for each of the capabilities that

the robot is required to meet. The subsystem then determines how well the robot meets the objective capabilities. Objective values are the user-defined ideal target capabilities. Both threshold and objective effectiveness values are output as a percentage of the robot's ability to meet each of these sets of capabilities. Finally, an average effectiveness is calculated. Average effectiveness is a hybrid measure between the threshold and objective values on which the model is visually steered. Average effectiveness is used to bridge the gap between a design being classified as infeasible because of not meeting the hard set minimum threshold values and the objective targets.

Effectiveness is the last of fifteen subsystems for which a detailed description was presented. The inputs and outputs for each subsystem were presented. Finally, the equations used in each subsystem and how each equation is related to another were described. Chapter 4 offers validation for each of these subsystems.

Chapter 4

Subsystem Validation

To validate the majority of the subsystem models discussed in Chapter 3, geometric and functional measurements of four robots were taken. These four robots include the Innovative Response Techologies, Inc. (IRT) BomBot®[1], Foster-Miller Talon®[2], Penn State Tankbot[3], and the REMOTEC, Inc. RONS[4]. Several of the measurements taken are inputs to the subsystem models while the remainder are outputs from the subsystems. The subsystem outputs were plotted against the measurements from the four robots, and the results are discussed in Chapter 4. A parameter with 100% agreement of model prediction to an actual measurement would lie on the 45 degree line in the figures.

4.1 Manipulator Subsystem Validation

The Manipulator subsystem is difficult to validate against any of the currently fielded robots because of the complexity of many of these designs. For example, the Talon has cameras attached to the manipulator. The Talon also has a humeral link that is not cylindrically shaped. It also has a prismatic joint within the forearm segment. The Talon utilizes brushless motors unlike the brushed motors modeled in this subsystem. It uses the same motor in the manipulator's elbow, manipulator's shoulder and drive train. The use of all common motors in the manipulator means that the designers did not design the manipulator the same way that this model is driven, i.e., the shoulder motor is not stronger than the elbow motor. Finally, it is

[1] http://www.irt-robotics.com/BomBot2EOD.htm
[2] http://foster-miller.qinetiq-na.com/lemming.htm
[3] http://controlfreaks.mne.psu.edu/previous_research.htm
[4] http://www.is.northropgrumman.com/by_solution/remote_platforms/index.html

impossible to know what lifting capacity the manipulator was designed for without running the risk of breaking the manipulator while testing. Therefore, at this time the manipulator subsystem has not been validated against an industrially available manipulator.

4.2 Batteries Subsystem Validation

The Batteries subsystem is validated against the batteries used in the four benchmarked robots. This subsystem was run by inputting the batteryType, busVoltage, and batteryUnitCapacity of the four robots. The mass and volume that were predicted by the subsystem for the batteries of each of the four robots were then plotted against the actual mass and volume of the corresponding batteries.

As can be seen in Figure 35 and Figure 36, as well as the corresponding Table 6 and Table 7, the model is very capable of predicting the mass and volume of a robot battery, respectively. The primary reason for error within the subsystem is due to numerous available packaging options that can affect the battery mass and volume. For example, not only can the battery be oriented in several different ways in order to change the length, width, and height of the vehicle, but some batteries may contain extra packaging in order to maintain the same form factor across multiple battery options. Additionally, the model does not account for additional components for extra protection of the batteries. This additional battery packaging may be part of the reason for the discrepancy between the Talon batteries and the design derived from the model. Further supporting this reasoning is that the BomBot has two different "sizes" of batteries available, yet their packages have the same dimensions and differ only in mass and power.

Another source of error in this validation is that the exact battery specifications have large variance for a given battery chemistry. For example, specific density for a lithium-ion battery can range anywhere from 200 to 280 [W-hr/m^3] [29]. Values for all of the chemistry specific constants described in the previous section were chosen in the middle of published ranges.

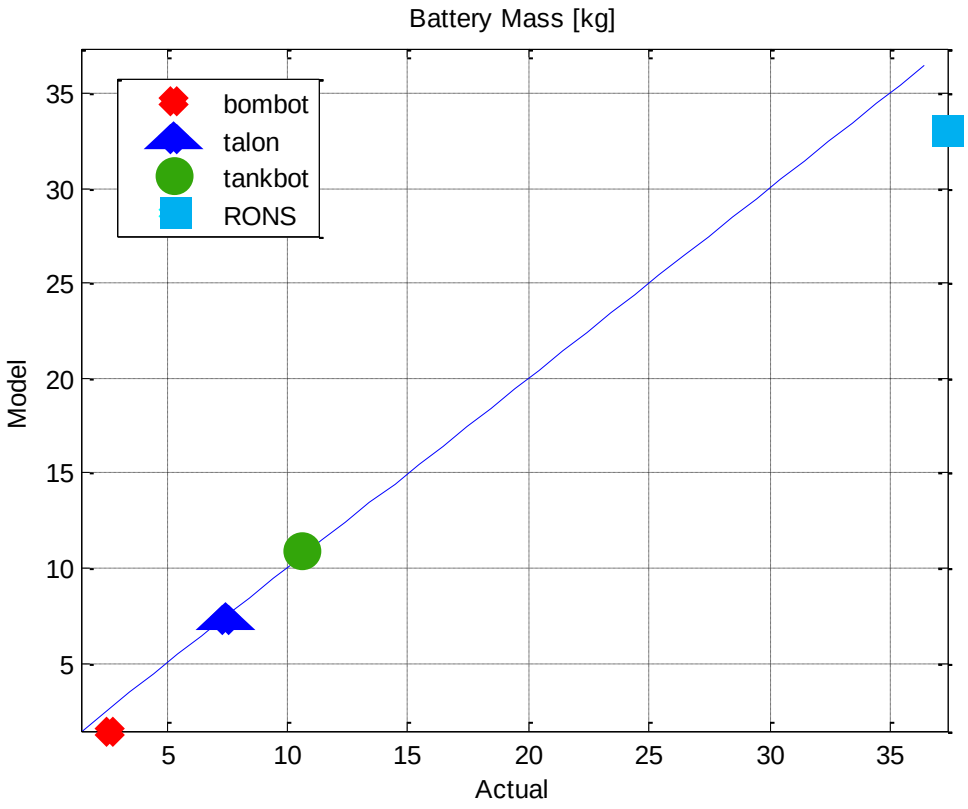

Figure 35: Comparison of Model Prediction to Actual Battery Mass

Table 6: Comparison of Model Prediction to Actual Battery Mass

	Actual Mass [kg]	Predicted Mass [kg]	Percent Error
Bombot	2.668	1.440	46.027
Talon	7.470	7.305	2.203
Tankbot	10.640	10.800	1.504
RONS	37.358	33.000	11.666

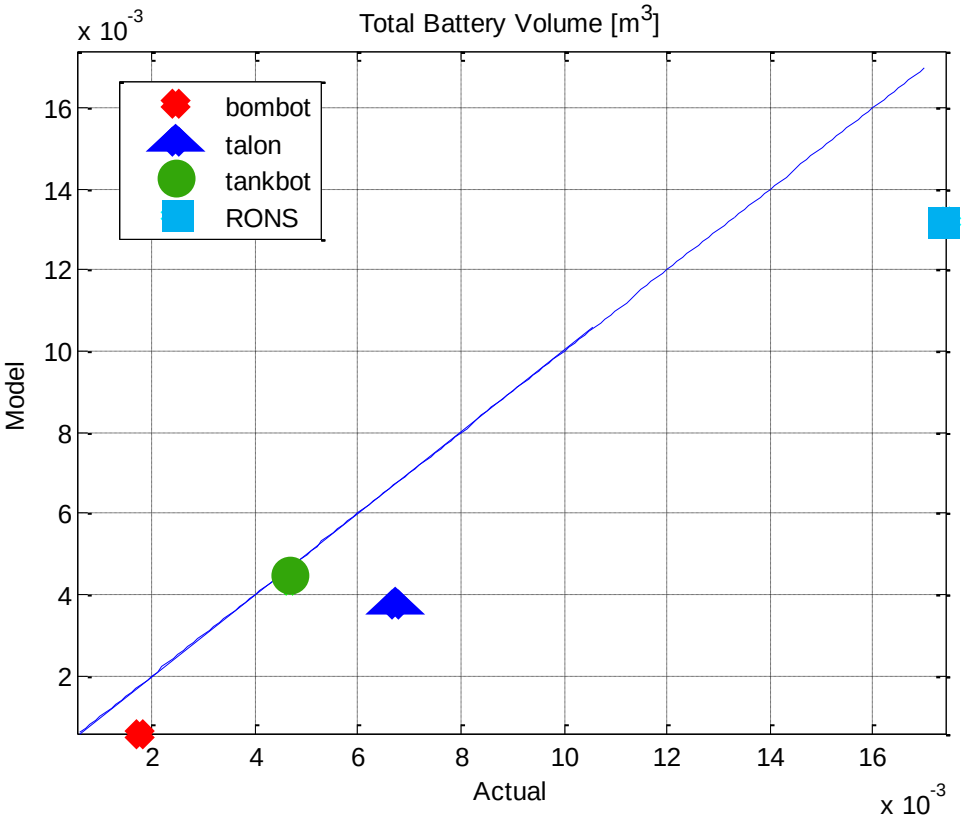

Figure 36: Comparison of Model Prediction to Actual Battery Volume

Table 7: Comparison of Model Prediction to Actual Battery Volume

	Actual Volume [m^3]	Predicted Volume [m^3]	Percent Error
Bombot	0.0018	0.0060	233.333
Talon	0.0067	0.0037	44.776
Tankbot	0.0047	0.0043	8.511
RONS	0.0174	0.0132	24.138

4.3 Motor Controller Subsystem Validation

Validation was not done on this subsystem because power into each motor controller was not measured during benchmarking thus leaving no voltage on which to scale the motor controllers.

4.4 Drive Motor Subsystem Validation

Four subsystem outputs of the Drive Motor subsystem were compared. The outputs compared are mass of each drive motor (see Figure 37 and Table 8), volume of the drive motor (see Figure 38 and Table 9), the gearbox ratio (see Figure 39 and Table 10), as well as the combined motor and gearbox mass (see Figure 40 and Table 11).

The primary factor contributing to the discrepancies in the model's conceptual designs and the actual robots is that the model consists of technologies that are different than what is used in the current robots and that technology is consistently improved over time. The primary discrepancy of the volume of the Talon motor is because the fit curves in the Simulink® model are based on brushed motor designs while the Talon uses brushless motors. The discrepancy of the mass and volume of the RONS is most likely due to technology improvements made over the years in motor packaging, power to mass ratio, etc.

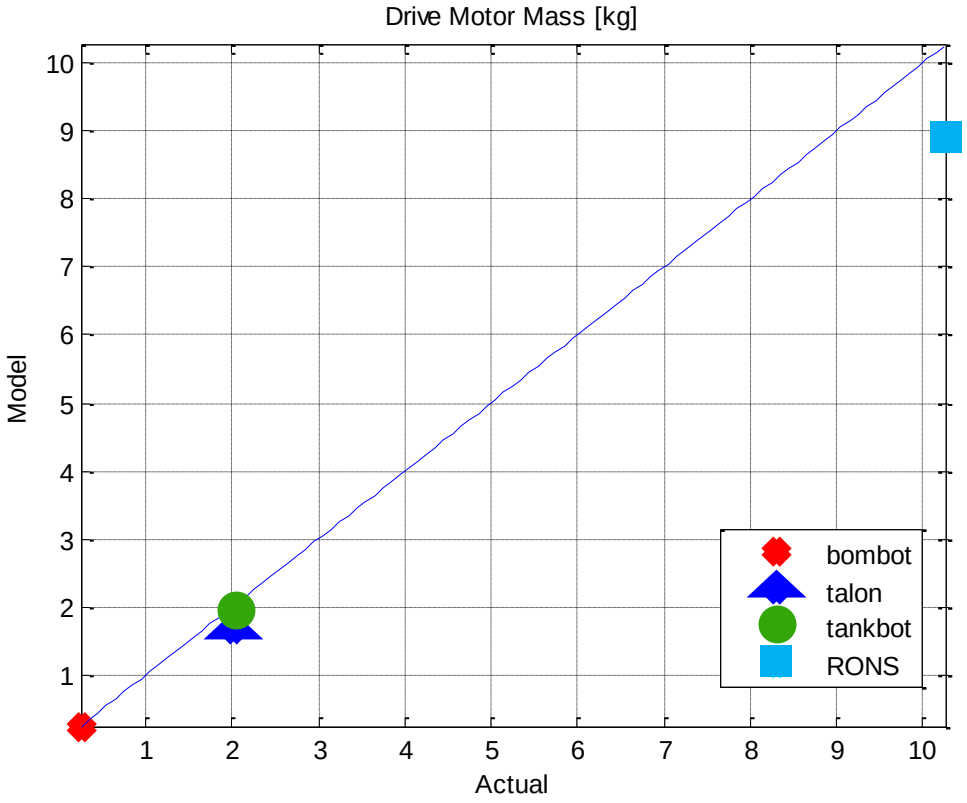

Figure 37: Comparison of Model Prediction to Actual Drive Motor Mass

Table 8: Comparison of Model Prediction to Actual Drive Motor Mass

	Actual Mass [kg]	Predicted Mass [kg]	Percent Error
Bombot	0.25	0.25	0.58
Talon	2.02	1.66	17.99
Tankbot	2.03	1.90	6.39
RONS	10.28	8.88	13.63

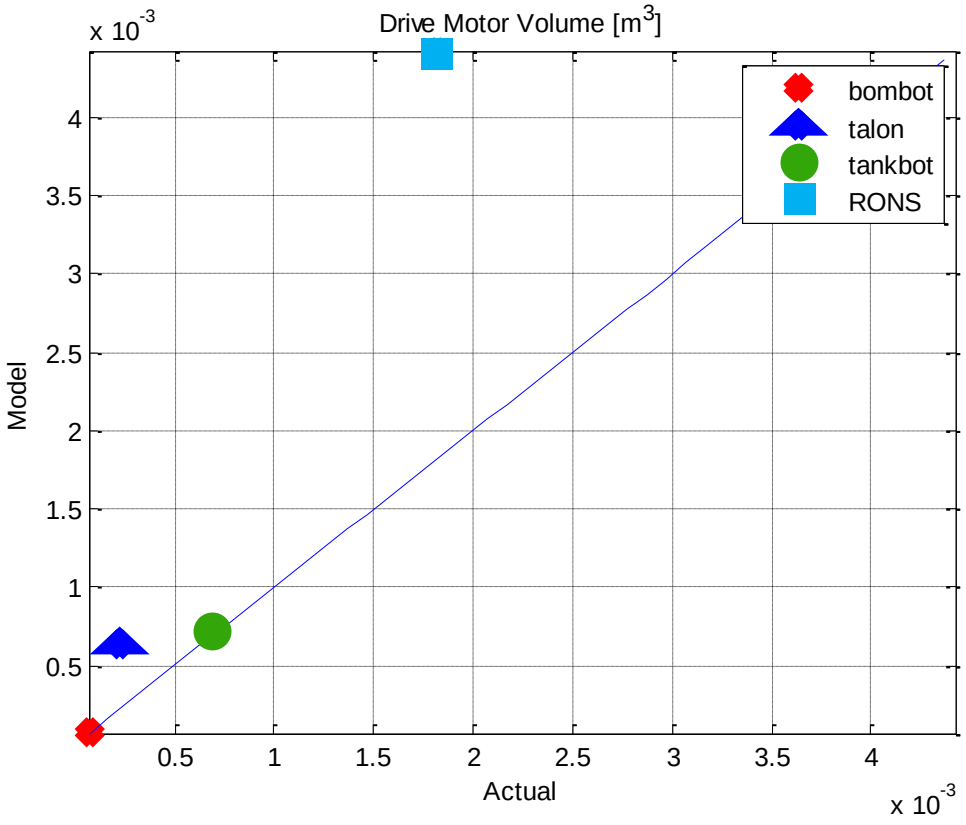

Figure 38: Comparison of Model Prediction to Actual Drive Motor Volume

Table 9: Comparison of Model Prediction to Actual Drive Motor Volume

	Actual Volume [cm^3]	Predicted Volume [m^3]	Percent Error
Bombot	68.28	69.04	1.12
Talon	227.12	629.06	176.97
Tankbot	693.99	737.49	6.27
RONS	1814.92	4423.06	143.71

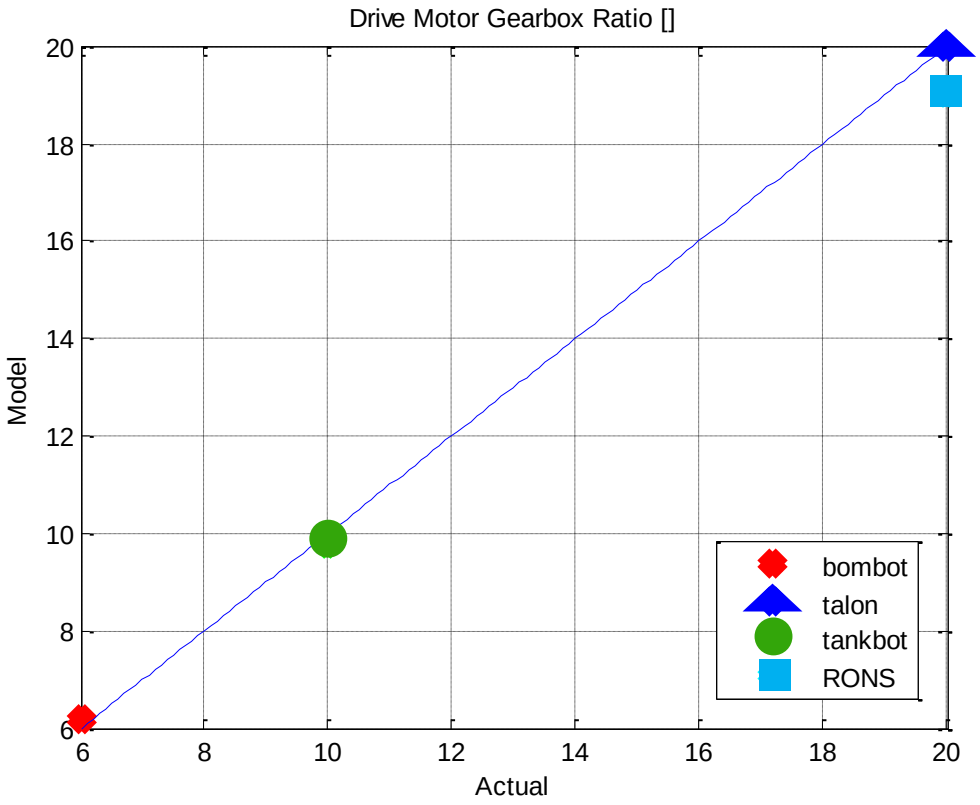

Figure 39: Comparison of Model Prediction to Actual Drive Motor Gearbox Ratio

Table 10: Comparison of Model Prediction to Actual Drive Motor Gearbox Ratio

	Actual Gear Ratio [ ]	Predicted Gear Ratio [ ]	Percent Error
Bombot	6.00	6.18	2.95
Talon	20.00	20.01	0.04
Tankbot	10.00	9.78	2.25
RONS	20.00	19.05	4.75

Validating the output value for the gearbox ratio is somewhat subjective. The Drive Motor subsystem sizes the gearbox so that the drive motor operates at peak efficiency. It is impossible to know for what speed the engineers of these four actual robots designed their gearboxes. Therefore, the input values for velocity were chosen such that the gearbox ratios

match. Once the gearbox ratios match, then the curve fit to calculate the mass of the gearbox was used and validated.

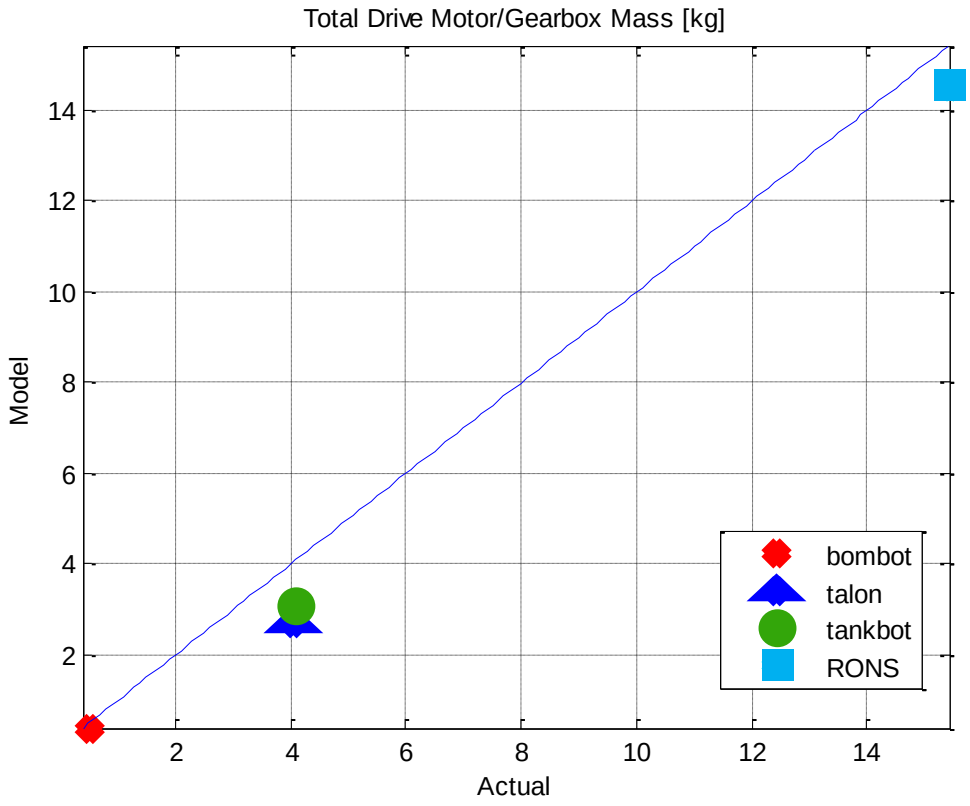

Figure 40: Comparison of Model Prediction to Actual Drive Motor and Gearbox Mass

Table 11: Comparison of Model Prediction to Actual Drive Motor and Gearbox Mass

	Actual Motor & Gearbox Mass [kg]	Predicted Motor & Gearbox [kg]	Percent Error
Bombot	0.50	0.39	21.22
Talon	4.05	2.72	32.75
Tankbot	4.06	3.04	25.15
RONS	15.42	14.53	5.79

4.5 Chassis Dimensions Subsystem Validation

All three of the Chassis Dimensions subsystem outputs were compared to actual measured dimensions. The three outputs compared are chassis length, chassis width, and vehicle length. Plots comparing the model predictions to the actual measurements are shown in Figure 41 through Figure 43. Tables comparing the same predictions and measurements can be found in Table 12 through Table 14.

The chassis' width has very good model to measurement agreement primarily due to the subjective payload dimensions. Payload dimensions were adjusted until there was a good model to measurement agreement of chassis width.

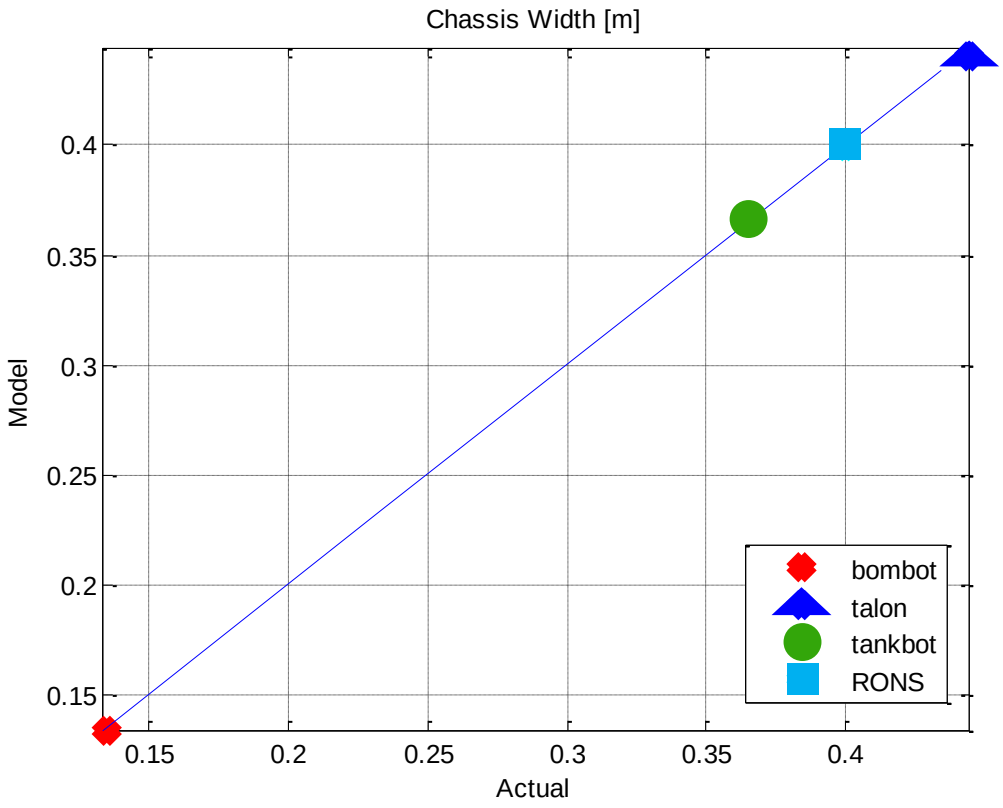

Figure 41: Comparison of Model Prediction to Actual Chassis Width

Table 12: Comparison of Model Prediction to Actual Chassis Width

	Actual Chassis Width [m]	Predicted Chassis Width [m]	Percent Error
Bombot	0.14	0.13	0.84
Talon	0.44	0.44	1.01
Tankbot	0.37	0.37	0.56
RONS	0.40	0.40	0.19

The lengths for the Talon and RONS are not as long as the model predicts them to be. The model first calculates width and then scales width to calculate length. Thus, if the same length to width ratio is not used for one of the benchmarked robots, then the value for length will exhibit error. Nonetheless, as shown in Table 13, the percent error is very small for the BomBot, Tankbot, and RONS. At just over 22%, the Talon is still within a reasonable tolerance of error.

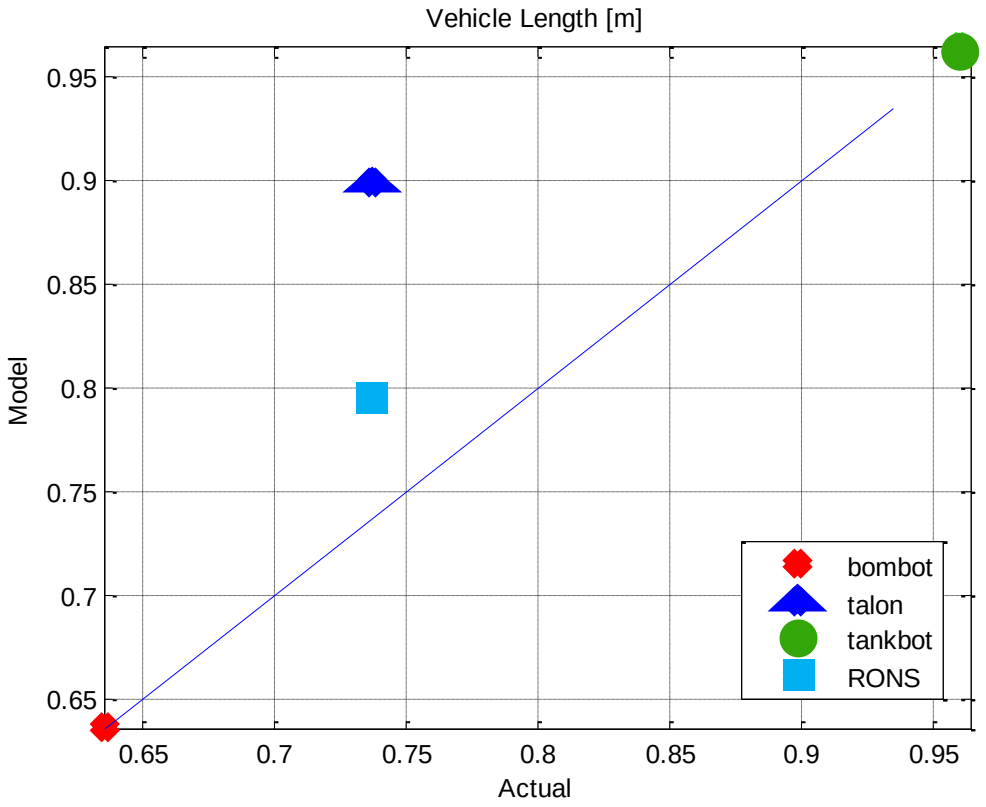

Figure 42: Comparison of Model Prediction to Actual Vehicle Length

Table 13: Comparison of Model Prediction to Actual Vehicle Length

	Actual Vehicle Length [m]	Predicted Vehicle Length [m]	Percent Error
Bombot	0.64	0.64	0.24
Talon	0.74	0.90	22.09
Tankbot	0.96	0.96	0.42
RONS	0.74	0.80	7.88

Chassis Height is difficult to validate for the Tankbot and RONS because it is difficult to identify exactly what the chassis is. The Tankbot and RONS are both very tall vehicles with large platforms sitting on top of what the model would define as a chassis structure. The chassis height was determined to be only the height of the bottom compartment. Additionally, there are

only three drive motor configurations defined within the model. The Tankbot and RONS drive motor configurations were approximated as similar but not equivalent configurations, thus explaining a probable reason for error.

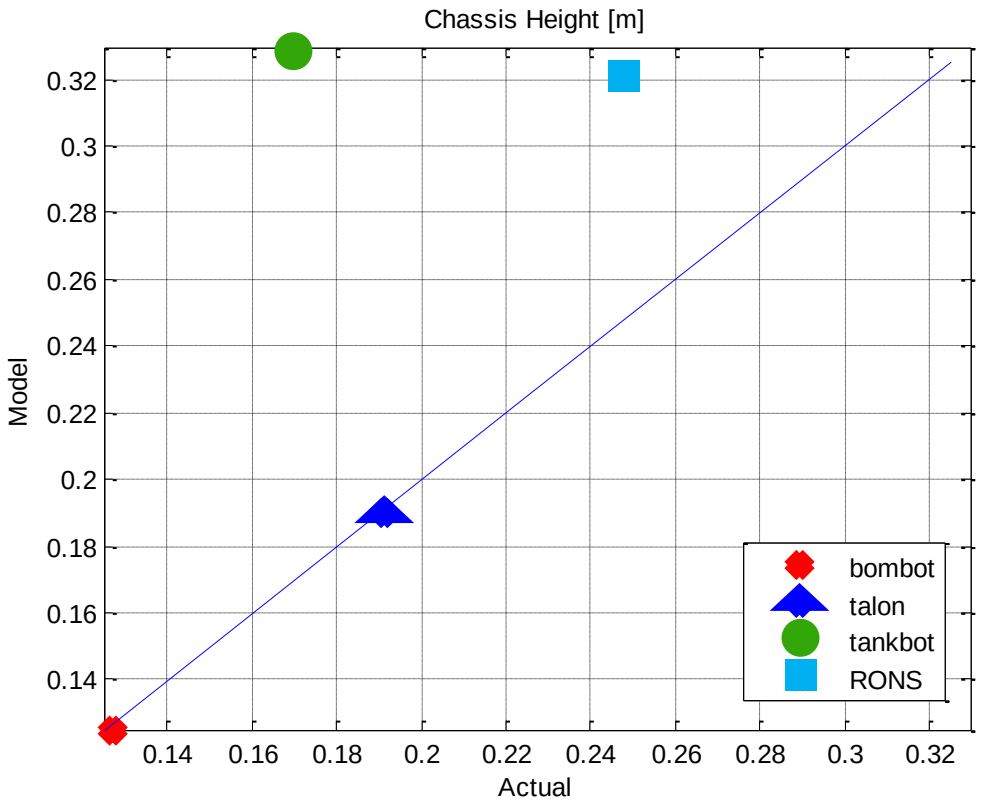

Figure 43: Comparison of Model Prediction to Actual Chassis Height

Table 14: Comparison of Model Prediction to Actual Chassis Height

	Actual Chassis Height [m]	Predicted Chassis Height [m]	Percent Error
Bombot	0.13	0.13	1.57
Talon	0.19	0.19	0.52
Tankbot	0.17	0.33	93.88
RONS	0.25	0.32	29.29

4.6 Chassis Structure Subsystem Validation

Calculating chassis mass is slightly ambiguous because it involves specifying a safety factor. It also assumes that the chassis is a rectangular aluminum slab which is not always the case in current robots. For example, the BomBot has a plastic structure that is not rectangular. The maximum payload capacity that the chassis was designed to be able carry is also not known. Nonetheless, it is possible to output correct chassis masses of all four actual robots as shown in Figure 2 and Table 15.

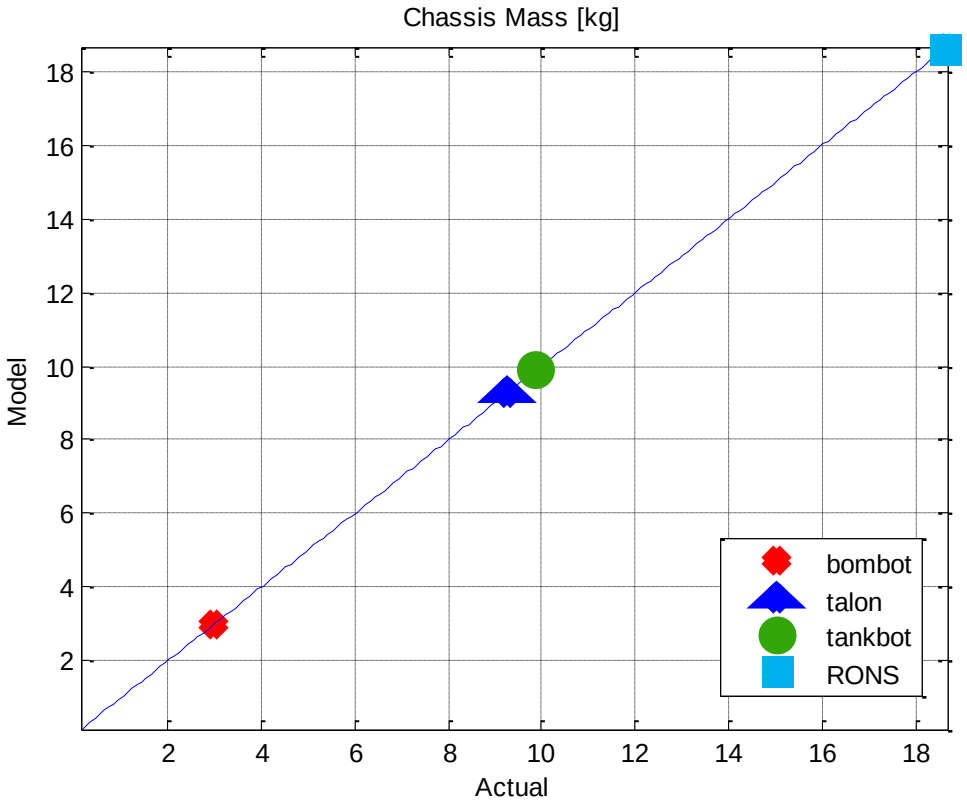

Figure 44: Comparison of Model Prediction to Actual Chassis Mass

Table 15: Comparison of Model Prediction to Actual Chassis Mass

	Actual Chassis Mass [kg]	Predicted Chassis Mass [kg]	Percent Error
Bombot	2.98	2.98	0.00
Talon	9.26	9.27	0.02
Tankbot	9.88	9.89	0.09
RONS	18.68	18.68	0.01

4.7 Wheels Subsystem Validation

Only one of the tested robots is a wheeled vehicle; the other three are tracked vehicles. Nonetheless, the same method of validation was used on this vehicle, namely the Bombot. Three output parameters were compared for this subsystem: wheelbase length, wheel width, and wheel mass.

The 39% error in the wheelbase length, seen in Table 16, is due to the fact that wheelbase length is calculated as the vehicle length minus the wheel diameter. However, the BomBot's wheels are not positioned at the very ends of its chassis as assumed in the model.

Table 16: Comparison of Model Prediction to Actual Wheelbase Length

	Actual Wheelbase Length [m]	Predicted Wheelbase Length [m]	Percent Error
Bombot	0.31	0.43	39.06

Because the wheel width is calculated as a function of ground pressure, the 32% error shown in Table 17 is associated with how the contact length is flagged, thus resulting in an increased wheel width. If the contact length is flagged when it is greater than 20% of the wheel diameter, instead of 40% as coded in the model, then the predicted wheel width matches the actual wheel with almost exactly (0.81% error). While this adjustment may be appropriate for smaller vehicles it may not be the case for larger vehicles with stiffer tires.

Table 17: Comparison of Model Prediction to Actual Wheel Width

	Actual Wheel Width [m]	Predicted Wheel Width [m]	Percent Error
Bombot	0.13	0.09	32.28

Error in the wheel mass, shown in Table 18, propagates from the error in the wheel width. Additionally, the value used for density is assumed to be a constant value of an average of experimental wheel and rim densities.

Table 18: Comparison of Model Prediction to Actual Wheel Mass

	Actual Wheel Mass [kg]	Predicted Wheel Mass [kg]	Percent Error
Bombot	2.99	3.47	15.83

4.8 Tracks Subsystem Validation

Validation of the tracks subsystem involved first finding an equivalent vehicle length that would output the appropriate length of the track in contact with the ground. The model calculates the wheelbase length as the vehicle length minus the wheel diameter. The model predicts a wheelbase length that looks similar to the Talon. Notice in Figure 45 that the tread of the track is the only thing separating the sprockets from the ground. However, the outermost sprockets on the Tankbot and the RONS do not come in contact with the ground (see Figure 45 for a photograph of the Tankbot's tracks), therefore, the length of the vehicle was adjusted to obtain an accurate basis for comparison for all of the other output parameters which depend on wheelbase length.

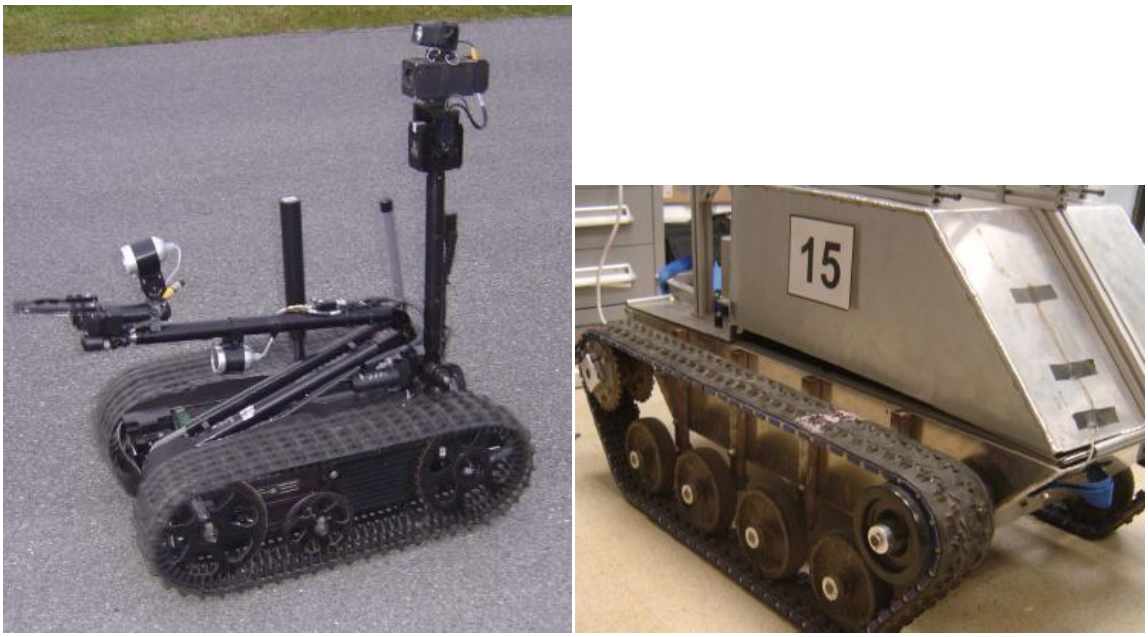

Figure 45: Talon (Left) and Tankbot (Right) Track Configurations

Once an accurate wheelbase length was determined, three model outputs were compared to the actual measurements for the three tracked robots. These three parameters are track width, track mass, and sprocket mass. The model predictions are compared to the actual measurements in Figure 46 through Figure 48 and Table 19 through Table 21 respectively.

The model predicted track width shows 100% agreement with the actual measurements, as shown in Figure 46 and Table 19. This agreement is due to the fact that width is scaled based on ground pressure as given by Equation 55; however, the ground pressure was calculated based on track width. Thus, this validation confirms that the ground pressure was initially accurately calculated, and it is correctly coded.

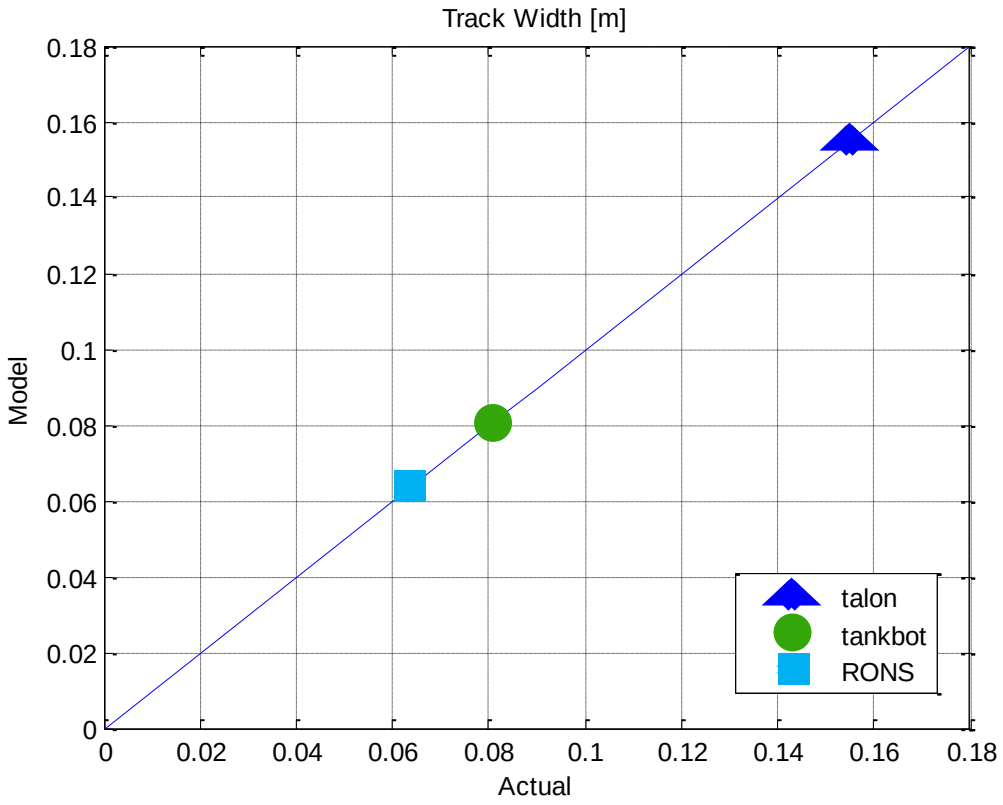

Figure 46: Comparison of Model Prediction to Actual Track Width

Table 19: Comparison of Model Prediction to Actual Track Width

	Actual Track Width [m]	Predicted Track Width [m]	Percent Error
Talon	0.16	0.16	0.00
Tankbot	0.08	0.08	0.00
RONS	0.06	0.06	0.00

Track mass for the Talon matches nearly exactly, as shown in Figure 47 and Table 20. This agreement is due to the scaling rule for mass, which is based on the Talon. However, the track mass of the Tankbot is significantly off due to a number of factors. First, the approximated thickness, which is based on the Talon, is less than half as thick as the measured value. Second, the material as well as the shape of the links of the tracks, are different than those of the Talon.

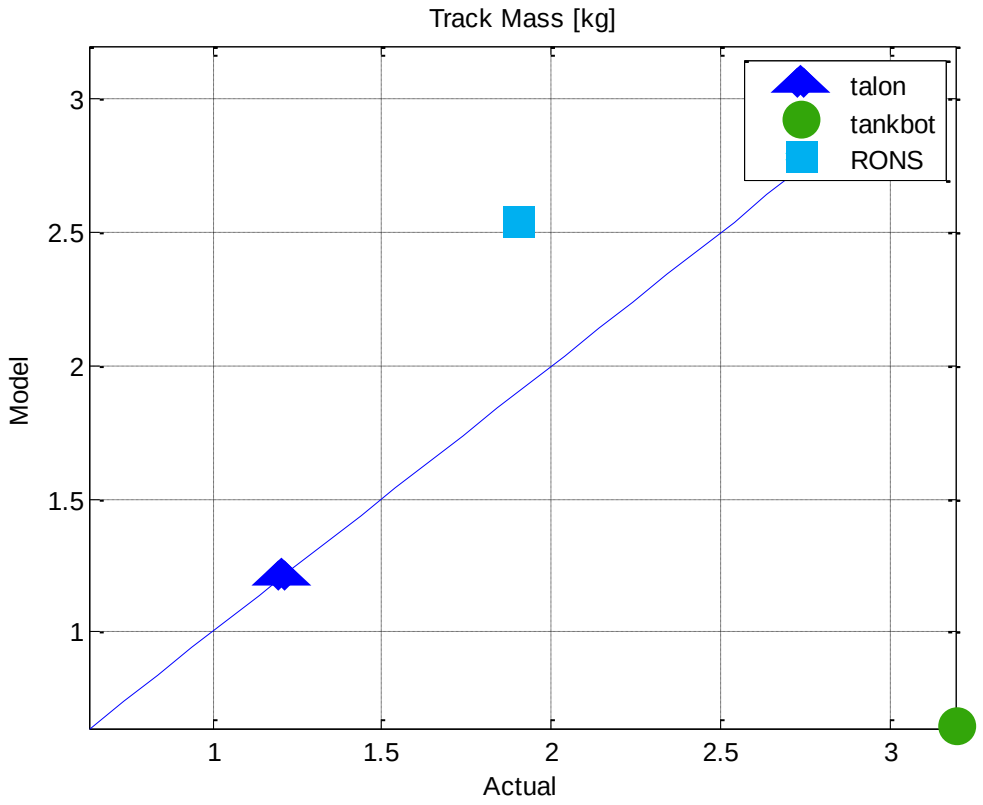

Figure 47: Comparison of Model Prediction to Actual Track Mass

Table 20: Comparison of Model Prediction to Actual Track Mass

	Actual Track Mass [kg]	Predicted Track Mass [kg]	Percent Error
Talon	1.20	1.21	0.96
Tankbot	3.20	0.64	80.03
RONS	1.91	2.54	33.38

The mass of the sprockets, as shown in Figure 48 and Table 21, show reasonable levels of agreement. The error seen in the sprocket mass of the Tankbot is likely due to the approximation of its material properties.

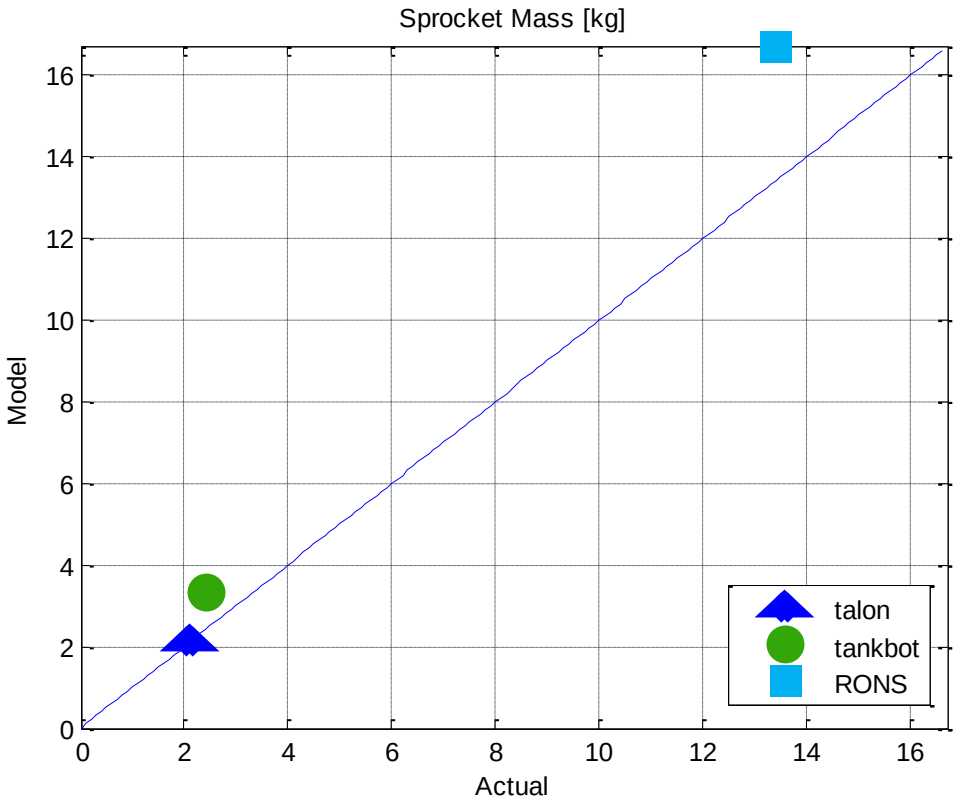

Figure 48: Comparison of Model Prediction to Actual Sprocket Mass

Table 21: Comparison of Model Prediction to Actual Sprocket Mass

	Actual Sprocket Mass [kg]	Predicted Sprocket Mass [kg]	Percent Error
Talon	2.10	2.12	0.81
Tankbot	2.39	3.29	37.75
RONS	13.43	16.70	24.36

4.9 Power Requirements Subsystem Validation

The calculation for maximum vehicle velocity is highly dependent on the user-defined auxiliary power as well as motor, gearbox and motor controller efficiency. However, many of these parameters are somewhat difficult to measure. For this reason, many of them were not used for benchmarking. Instead, the parameters previously mentioned were approximated using

curve fits and estimations of auxiliary power draw. These approximations and estimations led to a high level of agreement between the model predictions and the actual velocity measurements as shown in Figure 49 and Table 22.

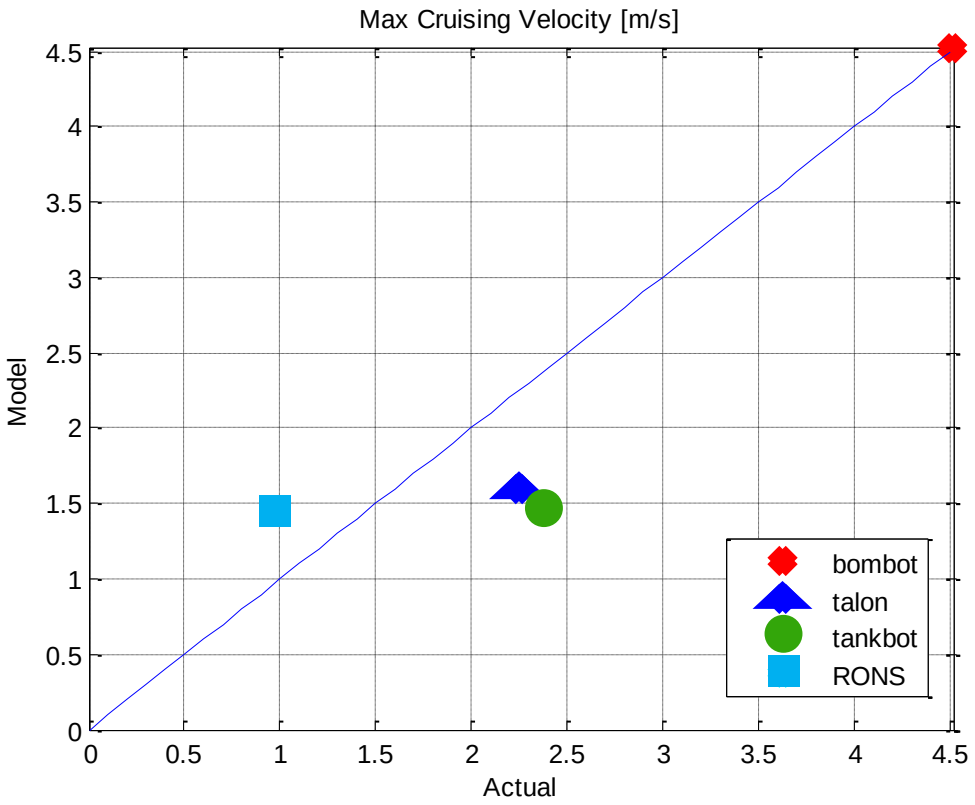

Figure 49: Comparison of Model Prediction to Actual Maximum Cruising Velocity

Table 22: Comparison of Model Prediction to Actual Maximum Cruising Velocity

	Actual Cruise Velocity [m/s]	Predicted Cruise Velocity [m/s]	Percent Error
Bombot	4.52	4.53	0.27
Talon	2.25	1.60	28.90
Tankbot	2.37	1.47	38.19
RONS	0.97	1.45	49.84

4.10 Endurance Subsystem Validation

As shown in Figure 50 and Table 24, the model very accurately predicts the maximum cruising time on a single battery charge. Additionally, as shown in Figure 51 and Table 24, the model accurately predicts the distance that the vehicle can drive on a single battery charge. The distance that the vehicle can drive is based on the duration of time that it can drive.

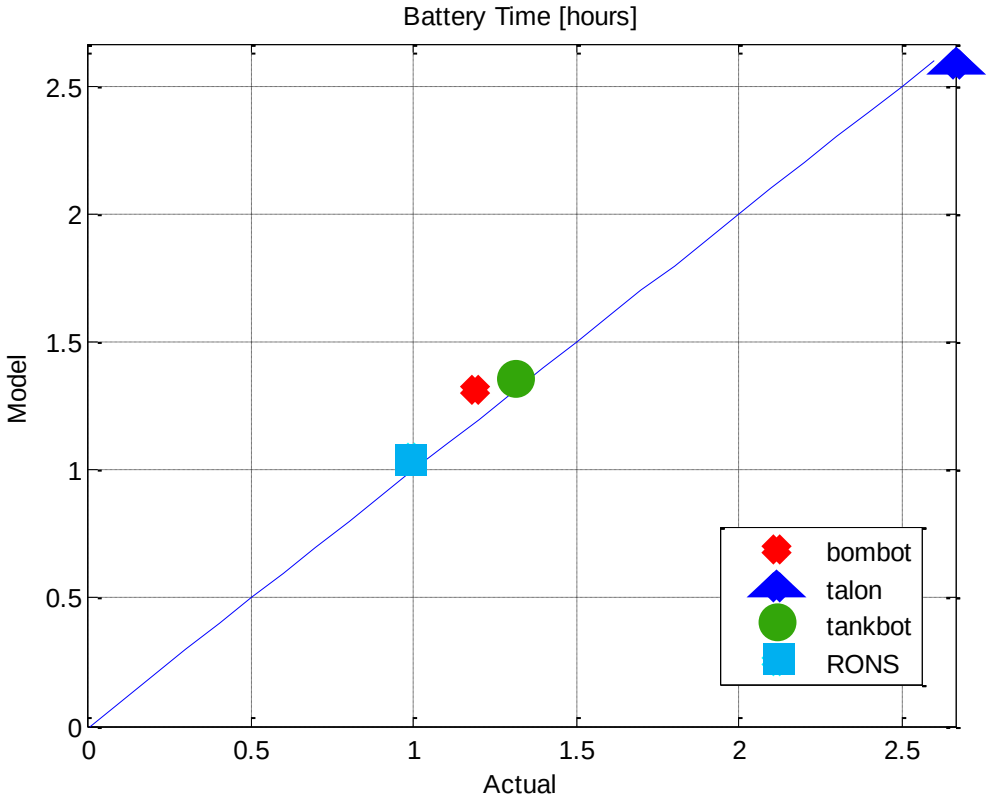

Figure 50: Comparison of Model Prediction to Actual Battery Time

Table 23: Comparison of Model Prediction to Actual Battery Time

	Actual Time [hrs]	Predicted Time [hrs]	Percent Error
Bombot	1.19	1.31	10.41
Talon	2.67	2.58	3.10
Tankbot	1.32	1.36	2.88
RONS	0.99	1.05	6.07

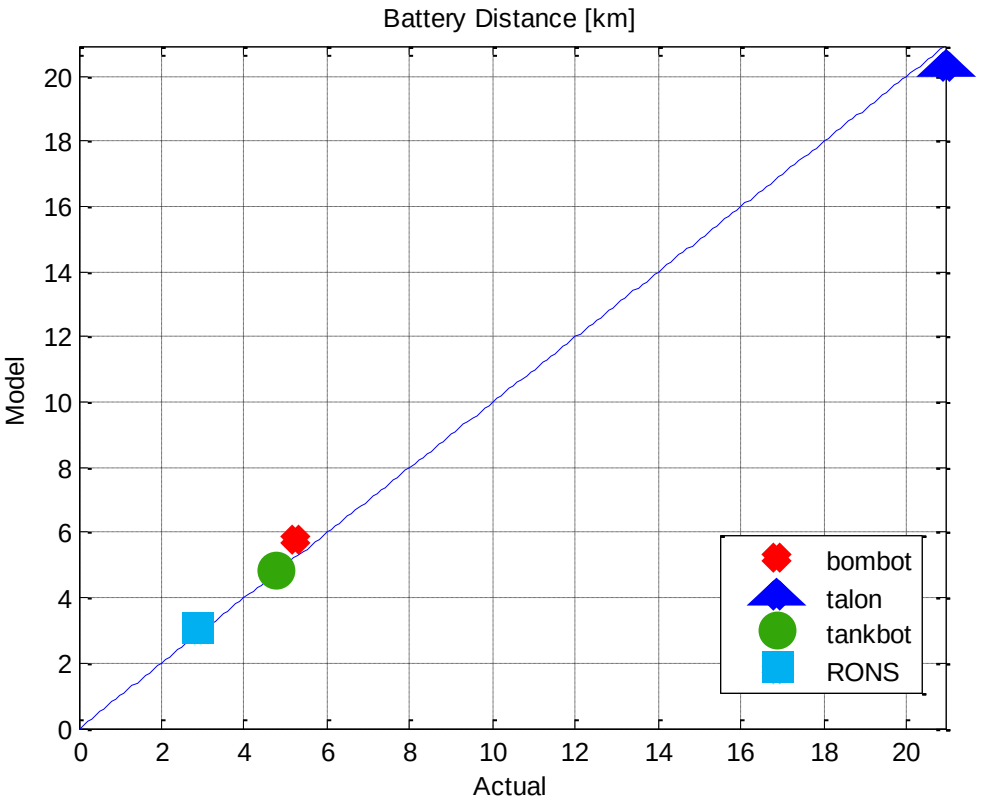

Figure 51: Comparison of Model Prediction to Actual Battery Distance

Table 24: Comparison of Model Prediction to Actual Battery Distance

	Actual Distance [km]	Predicted Distance [km]	Percent Error
Bombot	5.24	5.79	10.41
Talon	20.92	20.27	3.10
Tankbot	4.75	4.89	2.88
RONS	2.85	3.03	6.07

4.11 Total Vehicle Dimensions Subsystem Validation

As can be seen in Section 3.11.2, the Total Vehicle Dimensions subsystem model has a lot of assumptions. Yet as can be seen in Figure 52 through Figure 55 and

Table 25 through Table 28, these assumptions result in reasonably accurate vehicle dimensions and centers of gravity.

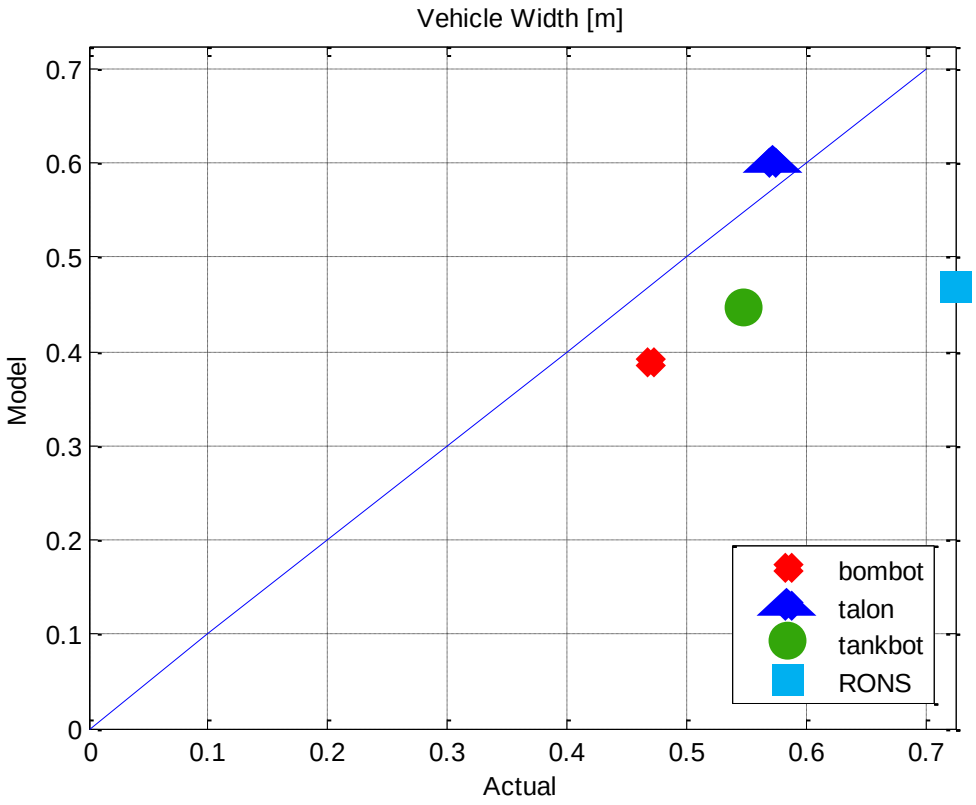

Figure 52: Comparison of Model Prediction to Actual Vehicle Width

Table 25: Comparison of Model Prediction to Actual Vehicle Width

	Actual Width [m]	Predicted Width [m]	Percent Error
Bombot	0.47	0.39	17.22
Talon	0.57	0.60	4.81
Tankbot	0.55	0.45	17.95
RONS	0.72	0.47	35.10

A percent error for CGySAE, shown in Figure 53 and Table 26 is not able to be calculated because CGySAE is always defined as zero, which would always result in an infinite percent error. The BomBot, Talon, and RONS all have a value for the center of gravity in the y-direction very close to the geometric center as assumed by the model.

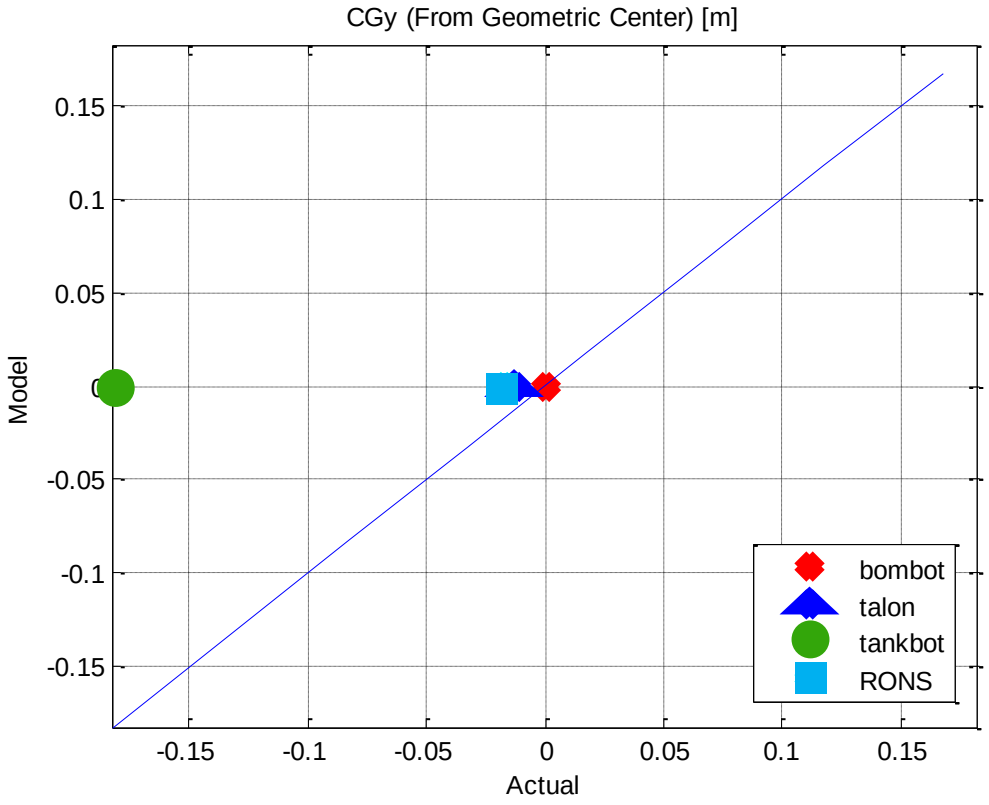

Figure 53: Comparison of Model Prediction to Actual CGySAE

Table 26: Comparison of Model Prediction to Actual CGySAE

	Actual CGy [m]	Predicted CGy [m]
Bombot	0.00	0.00
Talon	-0.01	0.00
Tankbot	-0.18	0.00
RONS	-0.02	0.00

The assumption is made that the distance from the rear axle to the center of gravity is half of the total vehicle length minus the wheel radius as described in Section 3.11.2. As shown in Figure 54 and Table 27, this is a reasonably valid approximation.

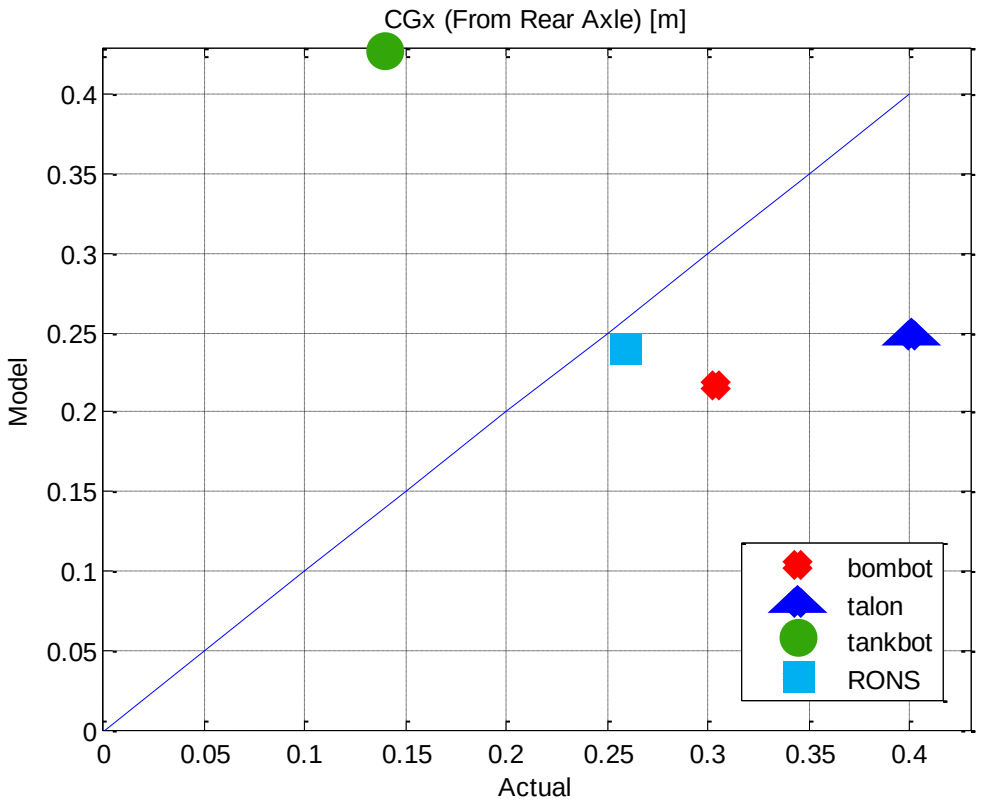

Figure 54: Comparison of Model Prediction to Actual CGxSAE

Table 27: Comparison of Model Prediction to Actual CGxSAE

	Actual CGx [m]	Predicted CGx [m]	Percent Error
Bombot	0.30	0.22	28.73
Talon	0.40	0.25	38.08
Tankbot	0.14	0.43	207.80
RONS	0.26	0.24	7.34

The disagreement shown in Figure 55 and Table 28 is likely due to any of the internal payload mass that is not located in the designated payload bay, which is positioned on the top of the chassis structure. Thus, if any computers or sensors are located in a higher up position than assumed in the model, as in the cases of the BomBot, Talon and Tankbot, the model under-predicts CGzSAE.

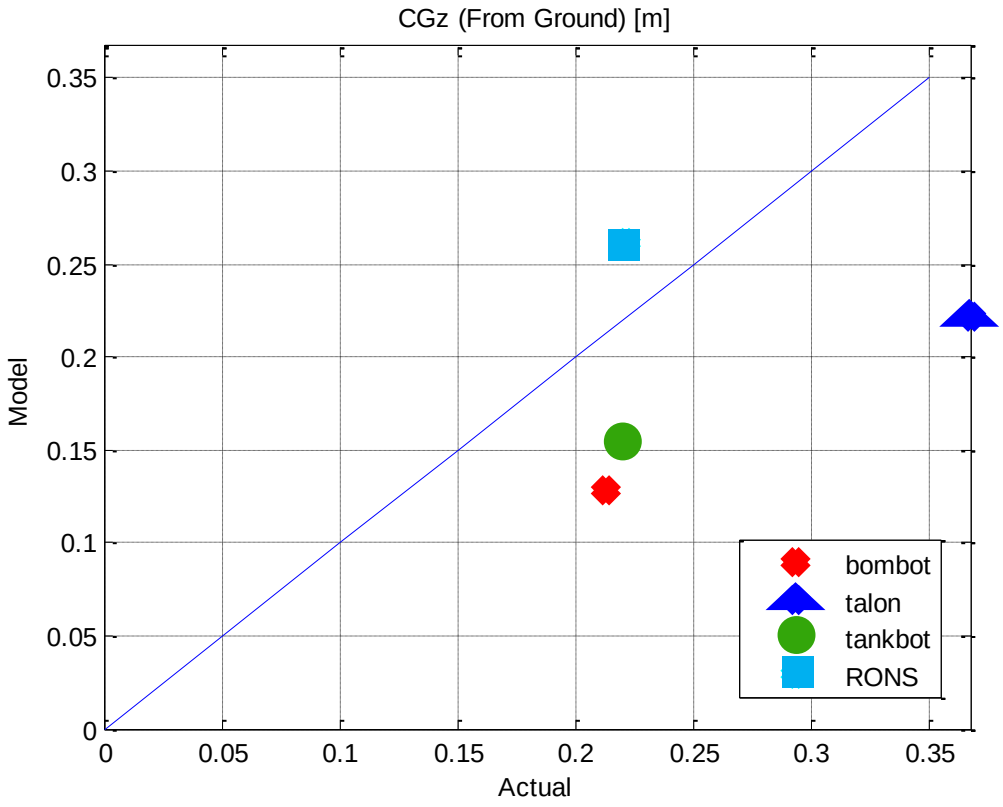

Figure 55: Comparison of Model Prediction to Actual CGzSAE

Table 28: Comparison of Model Prediction to Actual CGzSAE

	Actual CGz [m]	Predicted CGx [m]	Percent Error
Bombot	0.21	0.13	39.84
Talon	0.37	0.22	39.70
Tankbot	0.22	0.15	29.77
RONS	0.22	0.26	18.05

4.12 Functional Capabilities

Because the equations used in this subsystem are validated against the same four robots in another work [26], they are not validated here. The validation presented in this work shows that the equations used in the Functional Capabilities subsystem are very capable of accurately predicting capabilities.

4.13 Manipulator Capabilities

Like the Manipulator subsystem, the Manipulator Capabilities subsystem is difficult to validate against any of the currently fielded robots because of the complexity of many of these designs. Because of the numerous assumptions that would need to be made to validate this subsystem, the output values from the subsystem were only checked for feasibility.

4.14 Validation Summary

It was shown in this chapter that each of the subsystems presented in Chapter 3 is capable of accurately sizing components and predicting the capabilities of a robot. This validation was conducted by presenting comparisons of four existing robots to model representations of each of those robots.

The level of agreement in the model predictions to actual measurements is even more impressive when considering that the model is built on approximation after approximation. A small error due to one approximation would be multiplied by the small error due to another approximation. Over the course of the model, all of the small errors due to approximations would result in a very large error if not for a very detailed and rigorous validation process.

Additionally, this model is significantly more accurate than prior versions of the model. The previous version of this robot model, which was written in Mathematica®, often exhibited very large error. For example, the predicted cruising range was previously an entire order of magnitude off. However, due to the rigorous validation process described in this chapter and ensuring correct derivation of all of the equations used within the model, the cruising range now has at most a 10.5% error as shown in Table 24.

Chapter 5

Robot Parameter Trade Studies

5.1 Overview of Trade Studies

Tradeoffs exist between design parameters for any optimization problem. Increasing one parameter may inevitably cause another to decrease. Using ATSV, many of the competing design parameters existing within the model can be discovered. Some of these tradeoffs in robot design are obvious: an increase in close lifting capacity and manipulator length lead to an increase in total manipulator mass. Another example of an obvious tradeoff is an increase in manipulator mass or payload mass resulting in an increase in vehicle mass. However, many tradeoffs are not as obvious, and parallel coordinates plots in ATSV can be used to discover them.

Parallel coordinates plots are used to display multiple design variables for multiple design possibilities simultaneously [24]. These plots have been used in related work to display the same design variable for each product in a family simultaneously, thus making it easy to put the designer "in the loop" to make commonality decisions [23]. In the parallel coordinates plots shown in the subsequent sections, each vertical axis represents a design variable, and each horizontal line corresponds to a robot design. The location that a horizontal line intersects a vertical axis indicates the value that a particular design has for the given variable. The designs are colored based on the variable for which correlation is being explored as indicated by the legend on each plot. The designs on the high end of the variable being considered are red; the designs on the low end are blue. Strong correlations are found by looking for prismatic bands of color within the parallel coordinates plots.

5.2 Manipulator Tradeoffs

As mentioned in Section 5.1, there are several tradeoffs that are to be expected within the manipulator design variables. In these instances, ATSV is used to ensure that the outputs from the model make sense. The parallel coordinates plot shown in Figure 56 confirms that an increase in the close lifting capacity (2^{nd} axis) almost directly correlates with an increase in the total manipulator mass. There are other parameters that affect the total manipulator mass such as the total manipulator length (5^{th} axis), number of degrees of freedom (1^{st} axis), etc.; however, in general, a greater close lifting capacity translates to a greater manipulator mass.

It can also be seen in Figure 56 that the total manipulator mass is inversely proportional to the farthest distance that the robot can lift the close lifting capacity (4^{th} axis). This is because a heavier manipulator tends to tip the vehicle over when stretching horizontally. The mass of the vehicle and length of the chassis would both play into this inverse proportionality; however, as can be seen in Figure 56, the far reach distance is primarily dependent on the manipulator's mass.

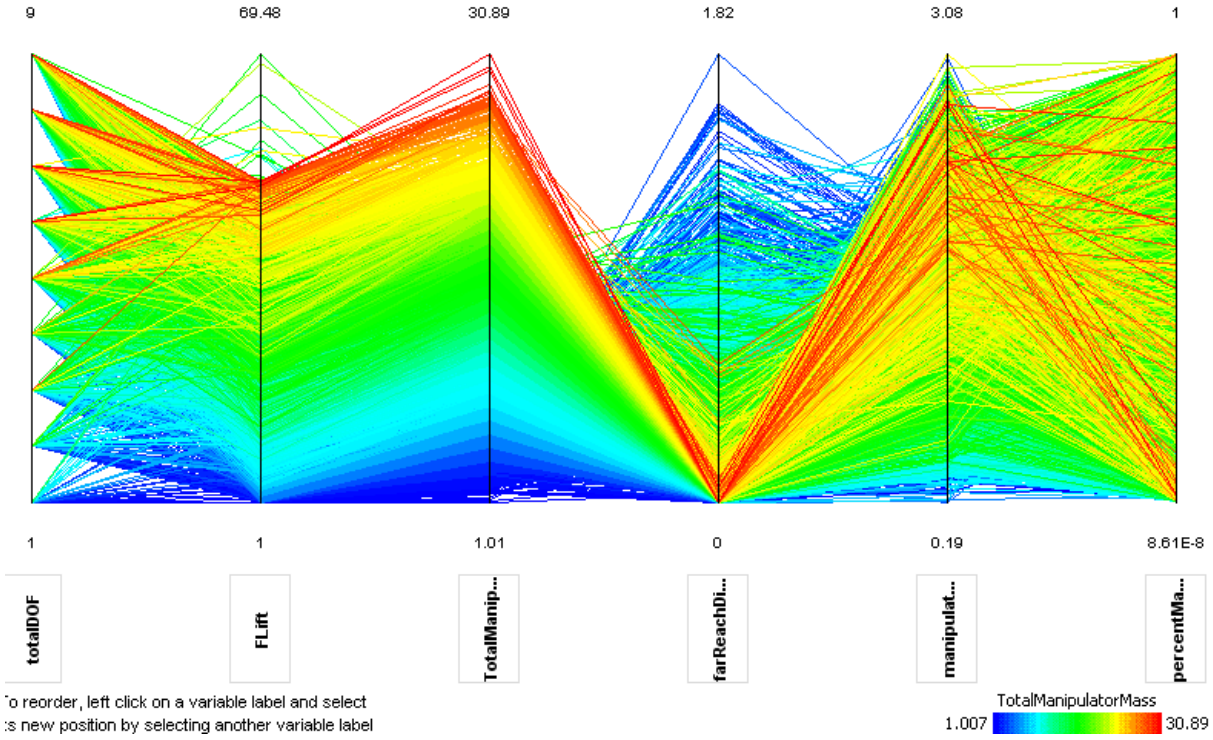

Figure 56: Parallel Coordinates of Manipulator Design Variables Colored on Total Mass

As shown in Figure 57, specifying a larger close lift capacity (2nd axis) yields electric motors with higher torques in the manipulator. Higher torques are also correlated with longer manipulator segments. Because the motors in the manipulator are scaled based on torque, manipulator motors that generate more torque also have higher total masses. This tradeoff can be seen in the third vertical axis of Figure 57. Similarly, a higher torque requires a stronger segment link. Thus, as shown in Figure 58, higher torques (3rd axis) result in larger outer segment radii.

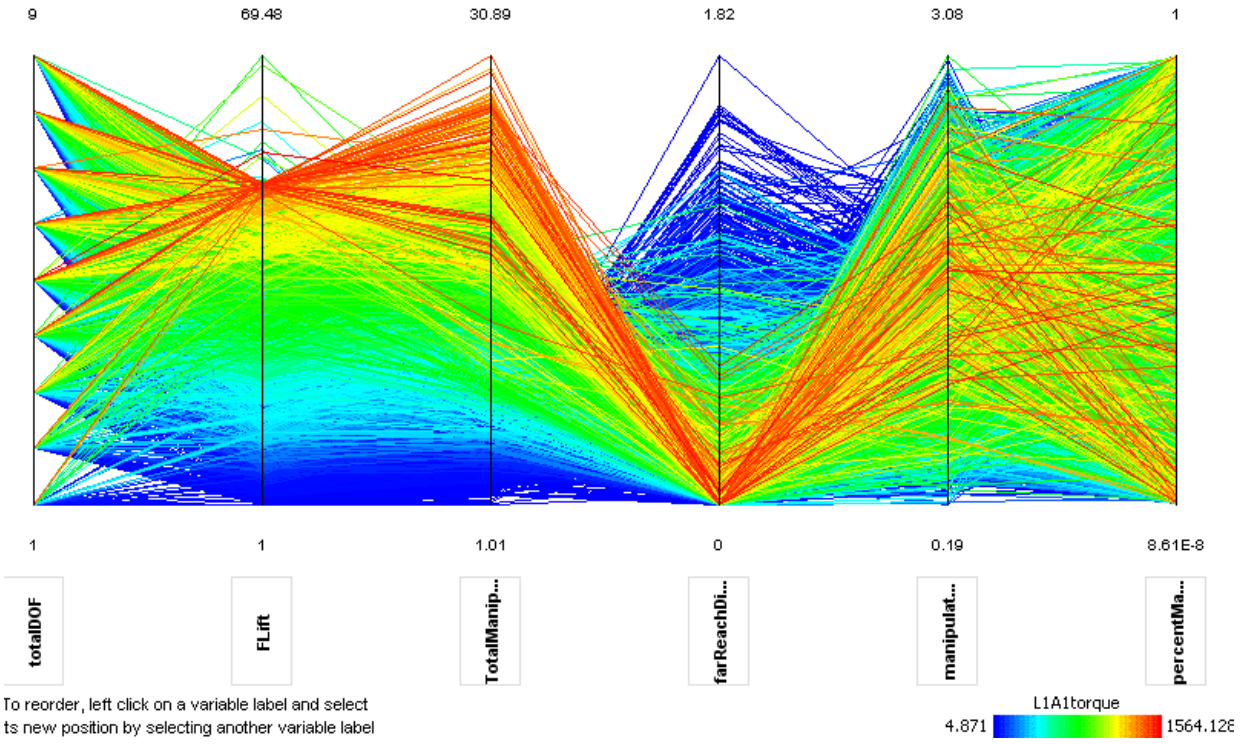

Figure 57: Parallel Coordinates of Manipulator Design Variables Colored on L1A1 Torque

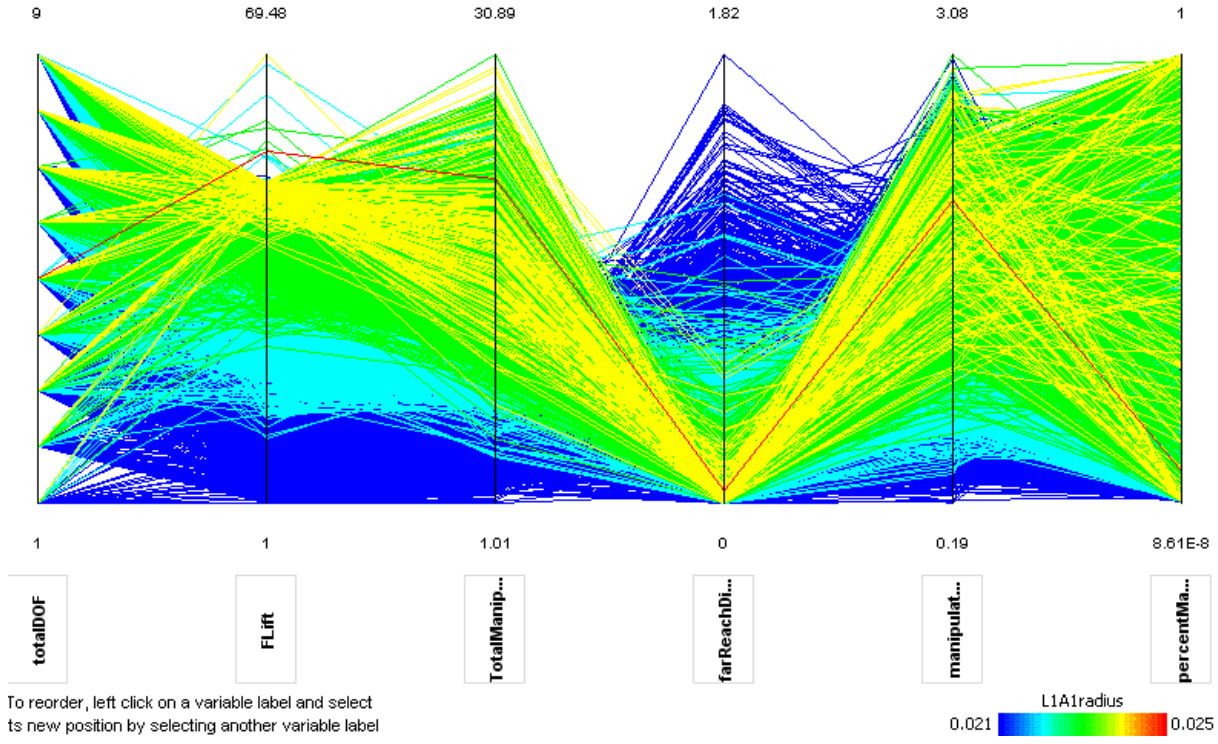

Figure 58: Parallel Coordinates of Manipulator Design Variables Colored on L1A1 Radius

As shown in Figure 59, a longer manipulator (5th axis) does not necessarily result in a longer far reach distance (4th axis). Rather, it can be seen from the parallel coordinates plot that an ideal balance must be found for a particular vehicle. Interestingly, the manipulators with lengths of approximately 1.5 [m] have the greatest far reach distance. A balance must be found because a longer manipulator not only weighs more but will be more prone to tipping before it can be fully extended.

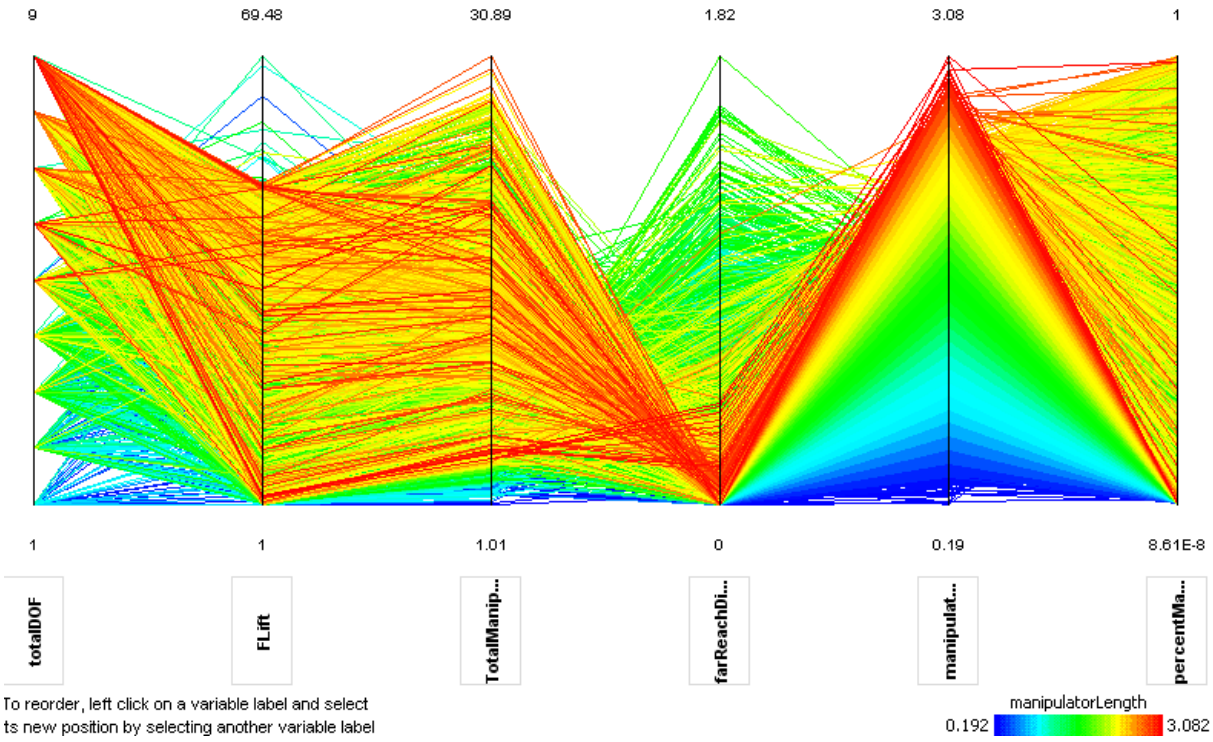

Figure 59: Parallel Coordinates of Manipulator Design Variables Colored on Total Length

Figure 60 shows a parallel coordinates plot of the same manipulator design variables as shown in Figure 56 through Figure 59. In this figure, the parallel coordinates plot is colored based on the manipulator's far reach distance. Additionally, the robots that are not able self-right in the event of a turtleback have been brushed out. The designs that are brushed out appear in

gray. Many of the designs with a greater far reach distance (4th axis) are not able to self-right. Interestingly, almost none of the robots with a far reach distance close to zero are brushed out; hence, a very large tradeoff is presented between the close lift capacity (2nd axis), far reach distance (4th axis and color) and self-righting capability (brushed designs). The robots that can most easily self-right typically have very heavy and strong manipulators in comparison to the rest of the robots. There are several robots that can self-right themselves that have large far reach distances; however, they have relatively small close lift capacities. There seems to exist an ideal range, or a "sweet spot" of close lift capacities that also have large far reach distances combined with self-righting capabilities.

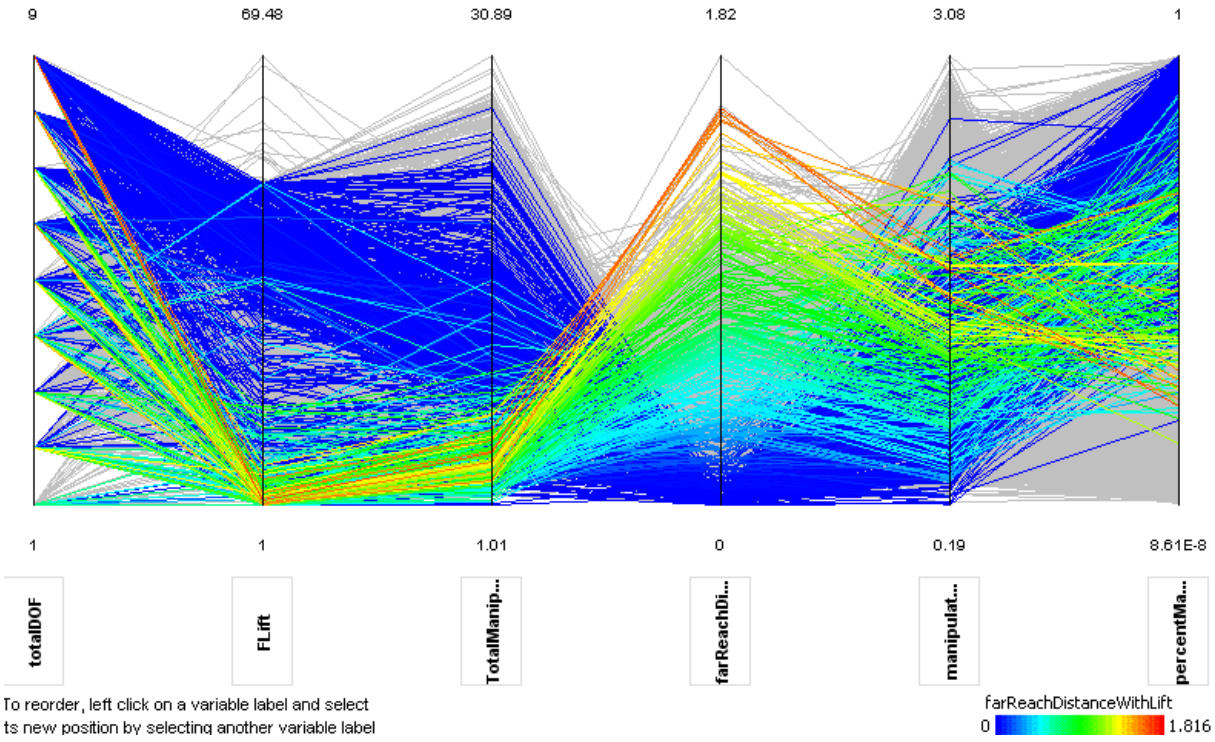

Figure 60: Parallel Coordinates of Manipulator Design Variables Colored on Far Reach

ATSV is also capable of producing two-dimensional scatter plots. Figure 61 shows a similar trend found in Figure 56, that is: increasing the close lifting capacity results in a heavier manipulator. Additionally, a steady decline in far reach distance can be seen in this plot for an increase in close lift capacity due to the total mass of the manipulator tipping the vehicle when fully extended. It can again be seen that there are several designs that seem to be able to reach farther and have more lifting capacity than the main cluster of designs shown in Figure 61. These manipulators are sized well for the chassis that they are attached to. All of these well-paired manipulators have a moderate total manipulator length that results in a moderate total mass.

To prevent ATSV from finding only one local minima, the search algorithm was stopped and started six times. On the last run, the upper bound on close lift capacity was increased to 70 [kg] thus explaining the cluster just under 50 [kg].

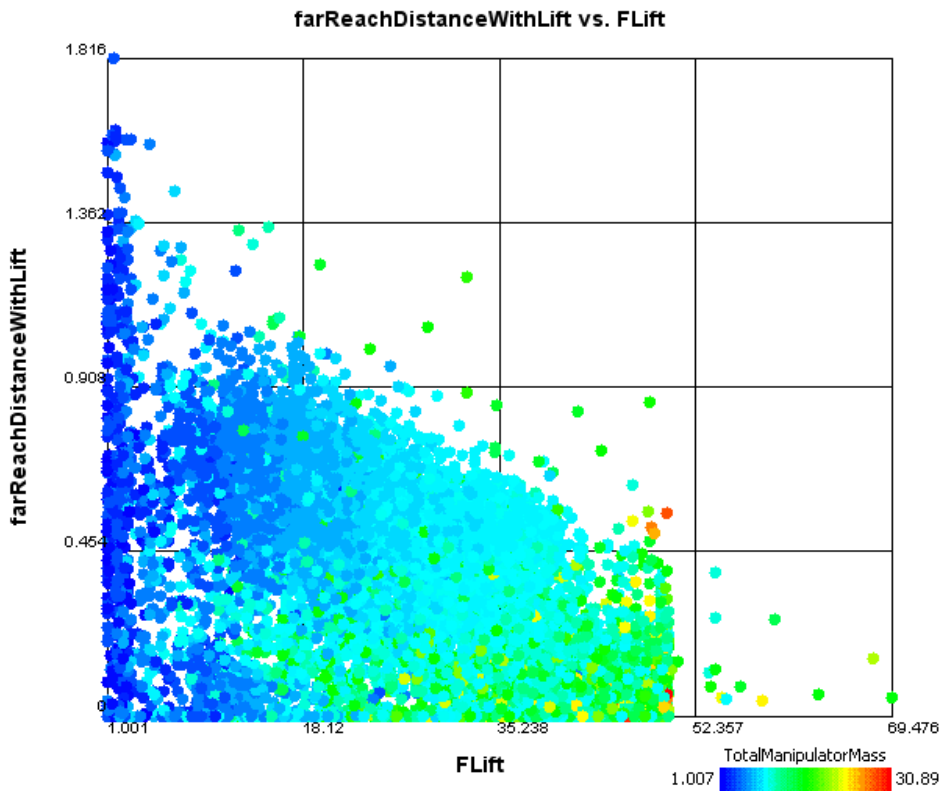

Figure 61: Scatter Plot of Far Reach Distance versus Close Lift Capacity

The robots that are not able to self-right can be brushed out of the scatter plot shown in Figure 61. The corresponding plot, in which only the robots that are able to self-right are shown in Figure 62. The majority of the designs with a self-righting capability are found in two locations: at very low close lift capacities or at far reach distances of zero (the robot would tip regardless of the location where it attempted to lift its close lift capacity). However, it can be seen that there are numerous designs that do not fall on either axis. The farther up and to the right in the plot that designs are, the better the tradeoff between close lift capacity and far reach distance while still retaining the self-righting capability.

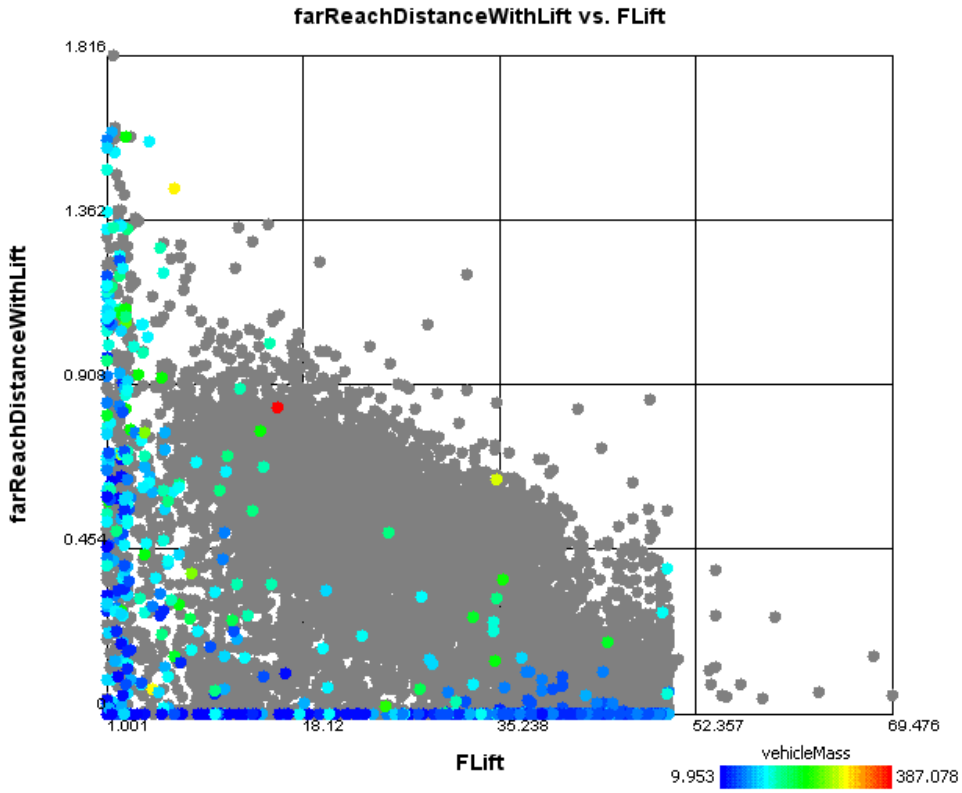

Figure 62: Far Reach Distance versus Close Lift Capacity Brushed on Self Righting

As seen in Equation 70, positioning the base of the manipulator closer to the robot's center of gravity in the x-direction, results in a greater ability to self-right. As shown in Figure 63, none of the over fifteen thousand designs with the manipulator positioned at the very front of the chassis is able to self-right. Rather most of the robots with the bases of their manipulators positioned just in front of their centers of gravity are able to self-right.

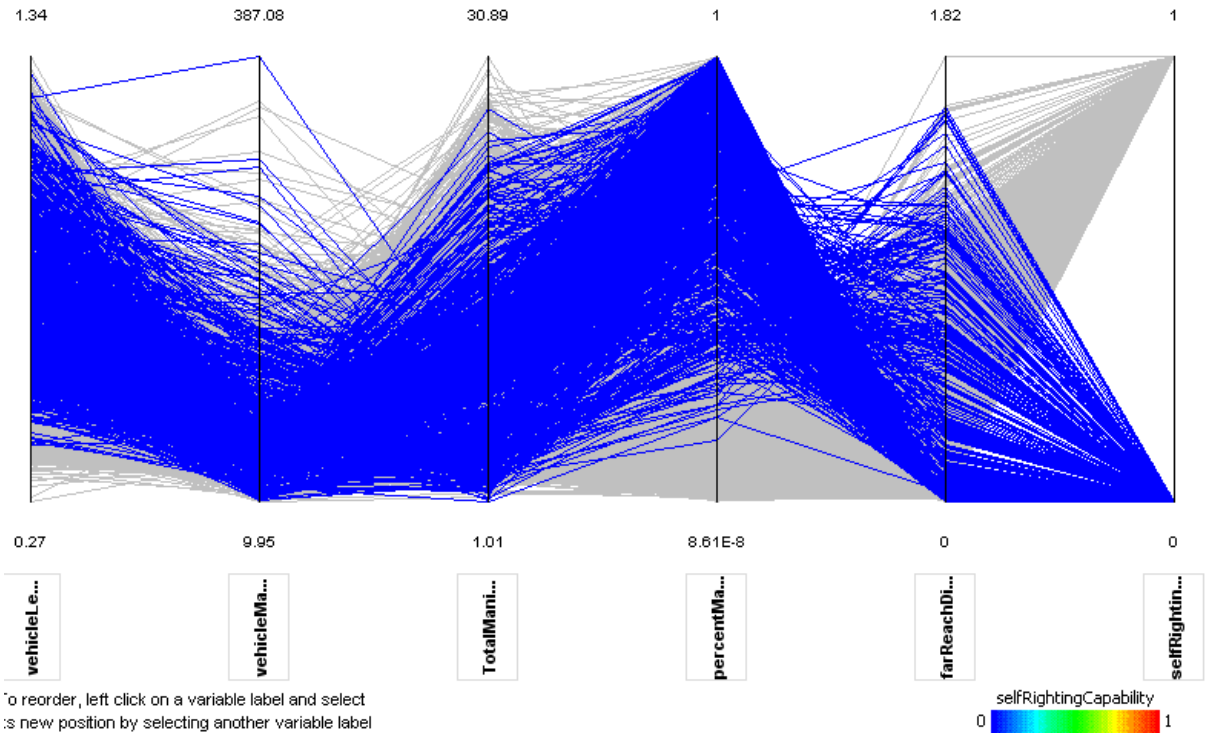

Figure 63: Parallel Coordinates of Manipulator Design Variables Brushed on Self Righting

5.3 Battery Tradeoffs

The tradeoffs of using an off-the-shelf battery such as the standard BB2590U versus a battery with no predetermined specifications were a question from the start of this project. It can be seen in Figure 64 that the custom batteries result in very large ranges of capacity (4^{th} axis), volume (5^{th} axis), mass (6^{th} column), and power (7^{th} column). Because of an upper limit of battery capacity, which is much higher than the fixed capacity of the BB2590U, these batteries can output much more power. This power is then used to size the drive motor. Therefore, Figure 64 shows a parallel coordinates plot for the battery parameters colored based on the drive motor mass. The first axis in this plot is battery type. The batteries of type 5 (custom Lithium Ion batteries) have a much larger range of specifications; thus, they can also support much larger

electric motors. This plot also shows that, as they should, volume, mass and power are all directly correlated with one another.

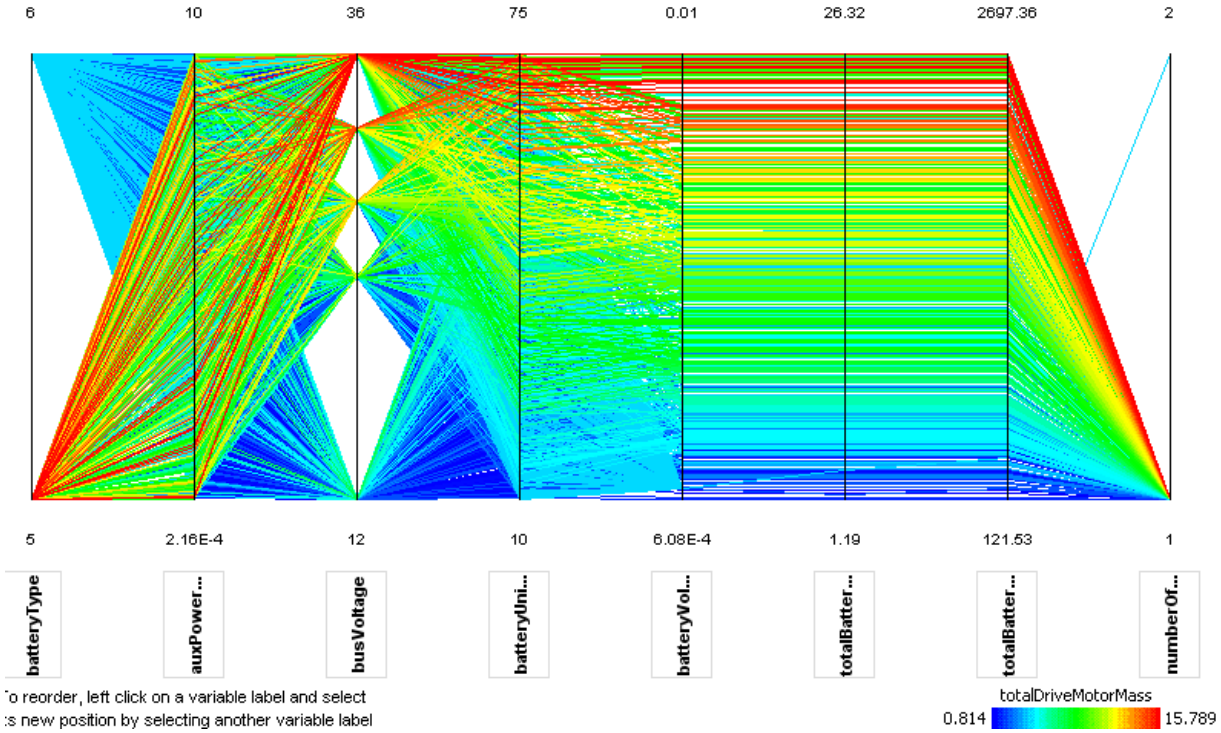

Figure 64: Parallel Coordinates of Battery Design Variables Colored on Drive Motor Mass

Heavier batteries result in greater power supplied to the electric drive motors. More power supplied to the electric drive motors results in heavier drive motors. Heavier batteries (6th axis) and heavier drive motors thus correlate with an overall heavier robot as shown in Figure 65.

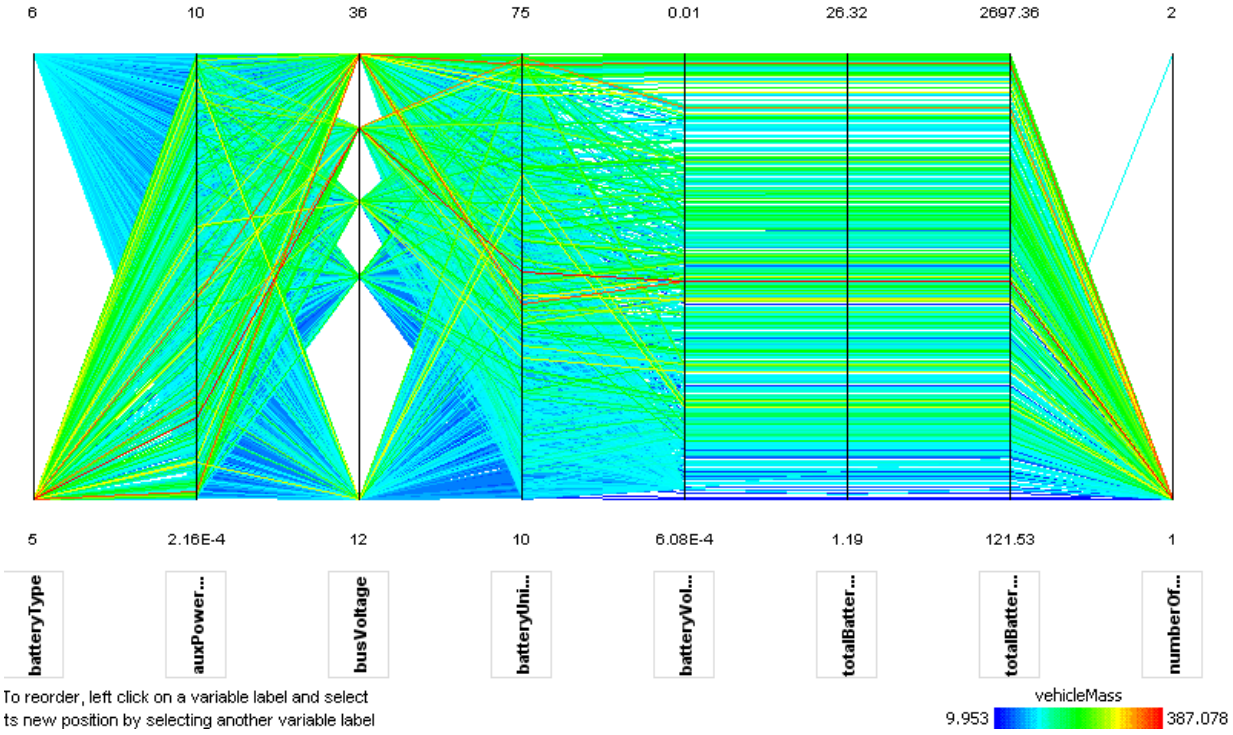

Figure 65: Parallel Coordinates of Battery Design Variables Colored on Vehicle Mass

Allocating more mass on the batteries and drive motors results in a robot with a higher maximum vehicle velocity. This tradeoff is shown in Figure 66. The heaviest batteries result in the largest total motor masses, which typically correspond to the most powerful electric motors that can typically drive the fastest.

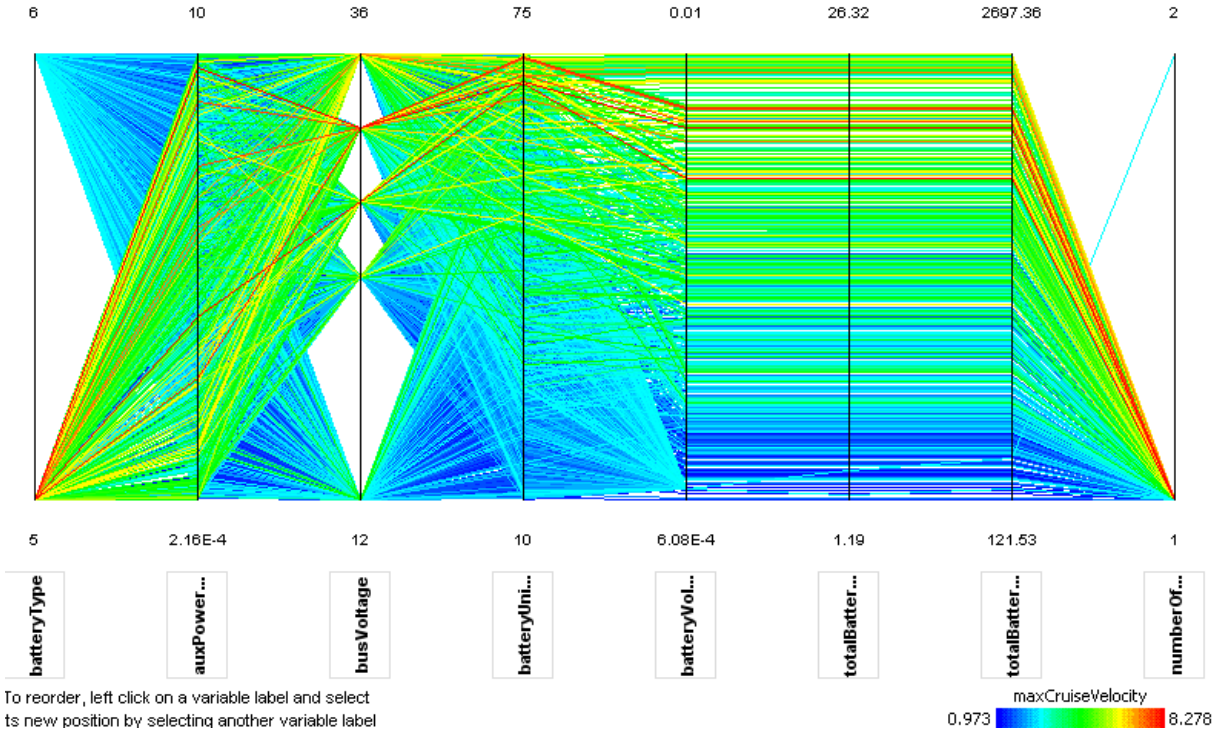

Figure 66: Parallel Coordinates of Battery Design Variables Colored on Maximum Velocity

The correlation of battery type to mass of the vehicle and maximum velocity is further explored in Figure 67. In this two-dimensional scatter plot, it can be seen that the BB2590U batteries (batteryType = 6) are confined to a much smaller area in the plot than the custom Lithium Ion batteries (batteryType = 5). The maximum velocity of a robot powered by a BB2590U is approximately 3 [m/s] while the maximum velocity of a robot powered by a custom Lithium Ion battery is approximately 8.3 [m/s]. The custom Lithium Ion batteries also lend themselves to higher overall vehicle masses. As shown in Figure 67, the maximum vehicle mass of a robot powered by two BB2590U batteries is approximately 110 [kg] while it is possible to power a 387 [kg] robot with a custom Lithium Ion battery. However, additional BB2590U batteries could be added to the 387 [kg] robot to provide the necessary driving power.

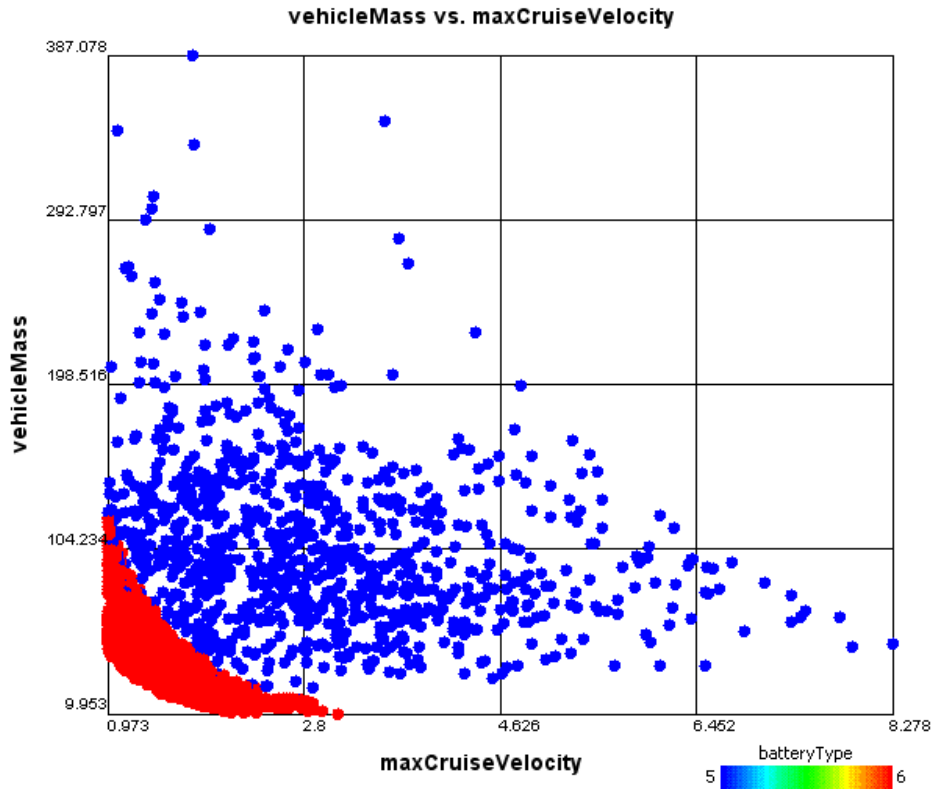

Figure 67: Vehicle Mass versus Maximum Velocity Highlighting Battery Type

While a robot powered by a BB2590U battery may not be able to drive as fast as one powered by a custom Lithium Ion battery, the robot powered by the BB2590U can drive for longer periods of time as shown in Figure 68. This can be explained by the modified Peukert number equation (see Equation 69). The lighter nature of these batteries result in lighter drive motors, lighter overall vehicle mass, and ultimately smaller resistive forces when driving. Therefore the electric current used in the modified Peukert number equation is smaller for a robot with a BB2590U battery, resulting in longer driving abilities.

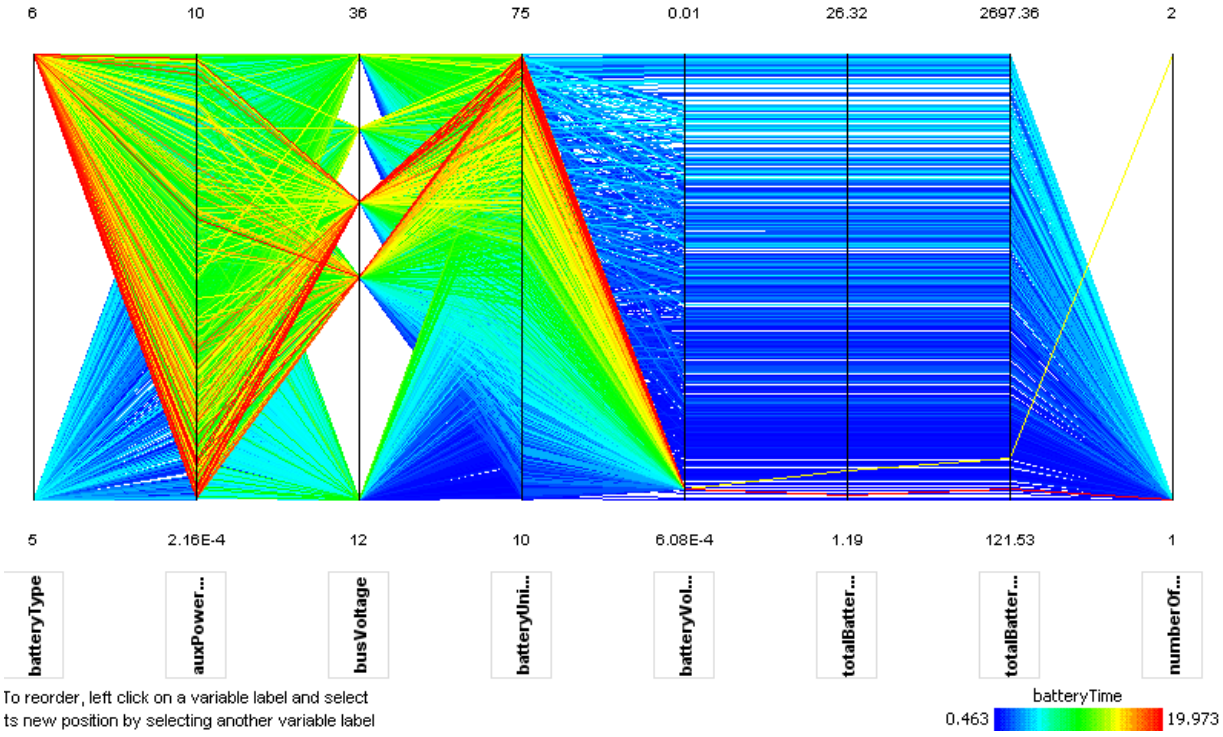

Figure 68: Parallel Coordinates of Battery Design Variables Colored on Endurance

5.4 Drive Motor Tradeoffs

As mentioned in Section 5.1, there are several tradeoffs that are not very intuitive that must be discovered using ATSV. Likewise, there are other tradeoffs that are to be expected. In these instances, ATSV is used to ensure that the model is generating accurate results. As would be expected, the electric drive motor length, diameter, and mass all increase proportionally to one another as shown in Figure 69.

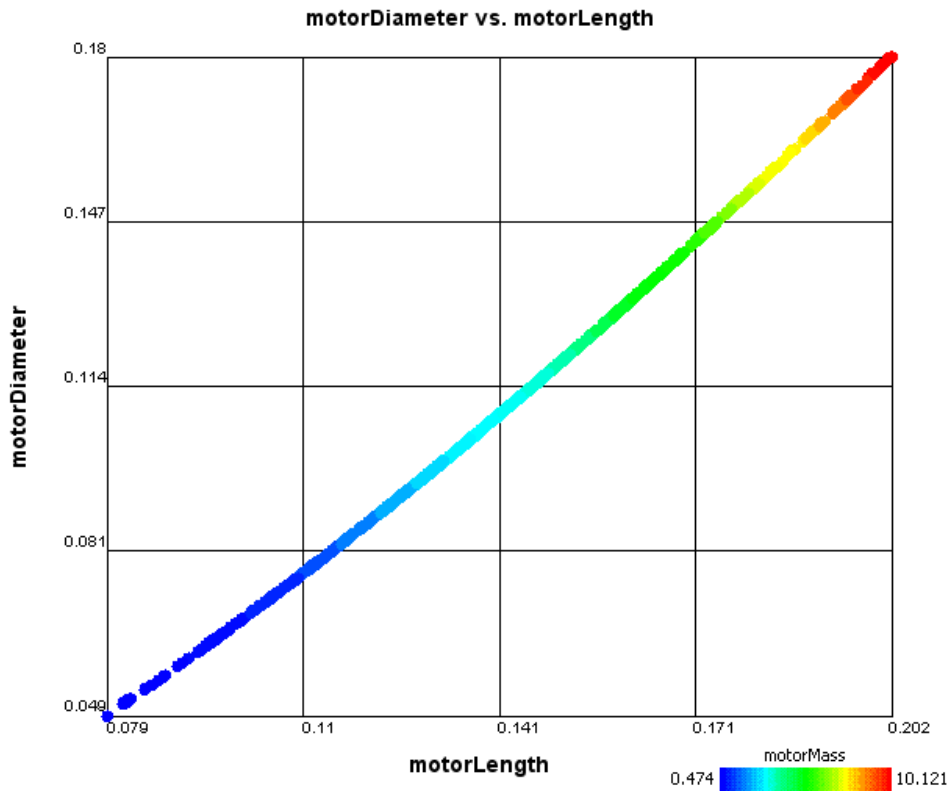

Figure 69: Drive Motor Diameter versus Length Colored on Mass

Once the proportionality of motor length, diameter, and mass has been confirmed it can be used to further examine tradeoffs within the drive motor design parameters. It has already been shown that higher battery capacities result in heavier drive motors because the model optimizes velocity for a given power supplied by the batteries. Figure 70 extends that tradeoff. As previously mentioned but shown in Figure 70 for completeness, larger drive motors result in vehicles with higher maximum velocities (7^{th} axis). Because the gearbox ratio (3^{rd} axis) is sized to allow the drive motor to spin at the maximum velocity some drive motors do not require gearboxes. Ultimately, there is a tradeoff between torque and velocity. The motors that do not use gearboxes, e.g., the vehicles with the highest maximum vehicle velocities, also have the least

amount of torque. A design with less torque may not be able to climb as steep of a slope, drag a very heavy mass, or skid steer.

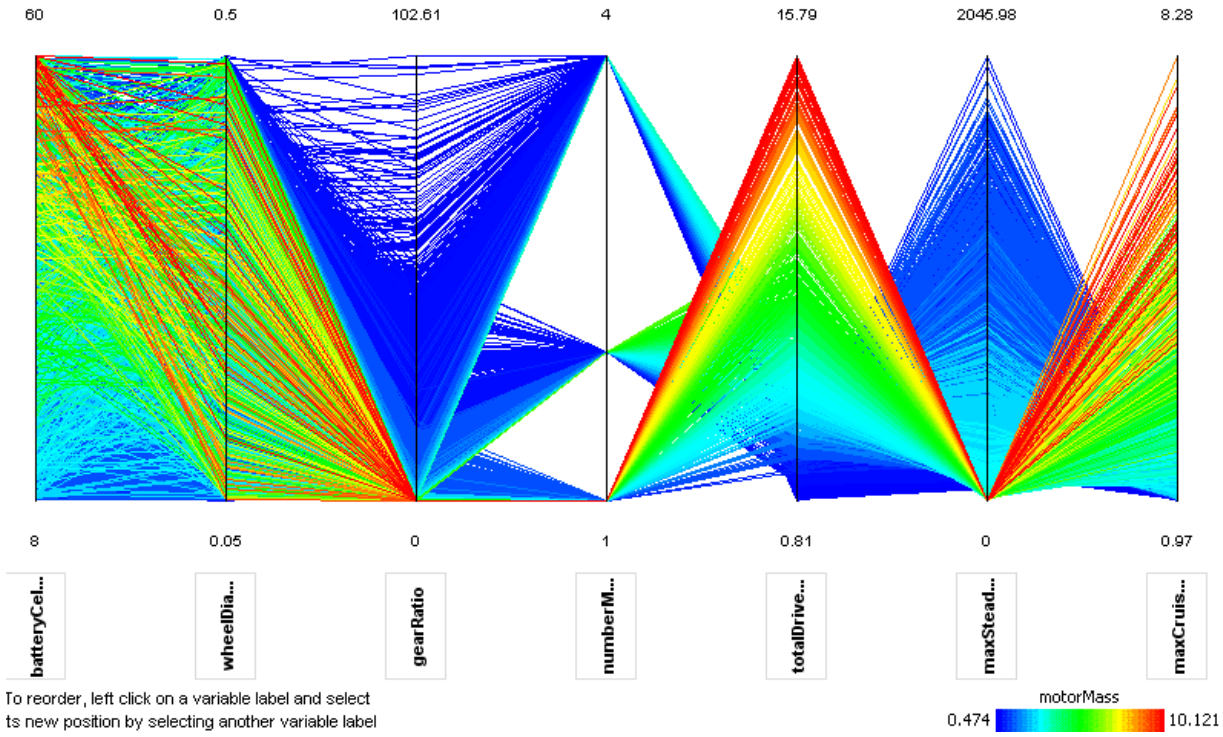

Figure 70: Parallel Coordinates of Motor Design Variables Colored on Motor Mass

Figure 71 shows the parallel coordinates plot of the drive motor design parameters colored based on the drive motor gearbox ratio. Here the effect of wheel diameter on gearbox ratio can be seen. As discussed in Section 3.4.2, the gearbox ratio is determined as a function of maximum velocity, continuous rotation speed of the drive motor, and wheel diameter. However, Figure 71 shows that the gearbox ratio is most strongly correlated with wheel diameter. It can again be seen that the total mass allocated to the batteries and thus the drive motors results in an increase in the maximum vehicle velocity.

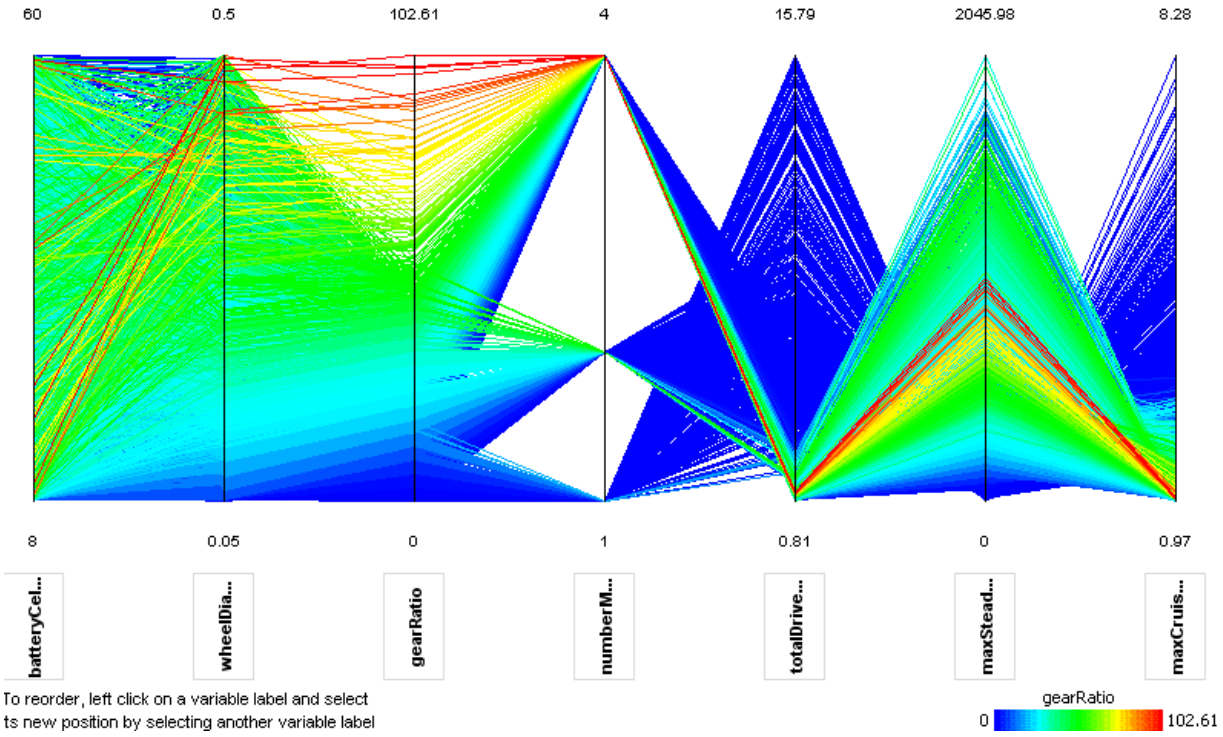

Figure 71: Parallel Coordinates of Motor Design Variables Colored on Gearbox Ratio

Figure 72 shows that a "sweet spot" exists within the drive motor parameters that optimizes the endurance of the vehicle. The vehicles that can drive for the longest amount of time have high battery capacities, medium wheel diameters, lower gearbox ratios, moderate continuous drive motor speeds, and lower maximum velocities.

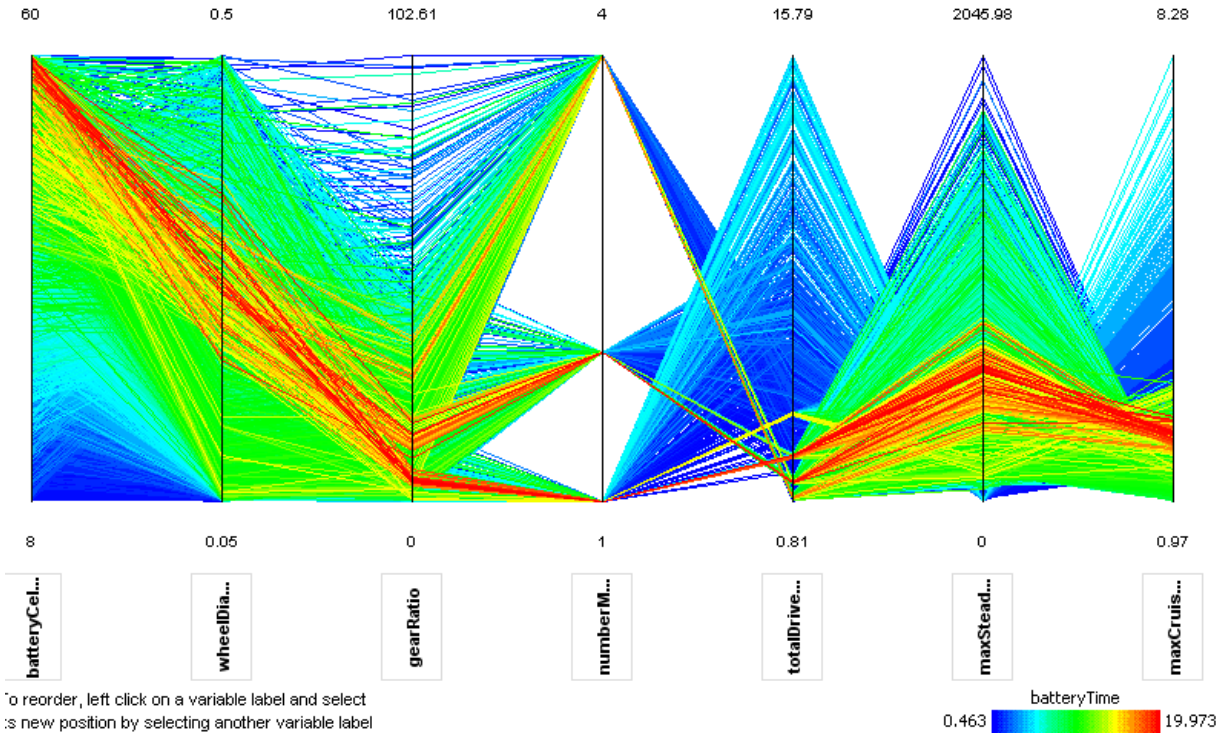

Figure 72: Parallel Coordinates of Motor Design Variables Colored on Endurance

5.5 Chassis Dimensions Tradeoffs

Figure 73 yields more insight into how payload dimensions are determined. Vehicle length (10^{th} axis) is primarily a function of chassis width (9^{th} axis) as shown in Figure 73. Because chassis width is calculated as the maximum of several parameters, payload width (2^{nd} axis) is sometimes directly correlated with the chassis width (9^{th} axis). However, it is sometimes not at all correlated, which explains the sporadic red lines crossing the 2^{nd} axis.

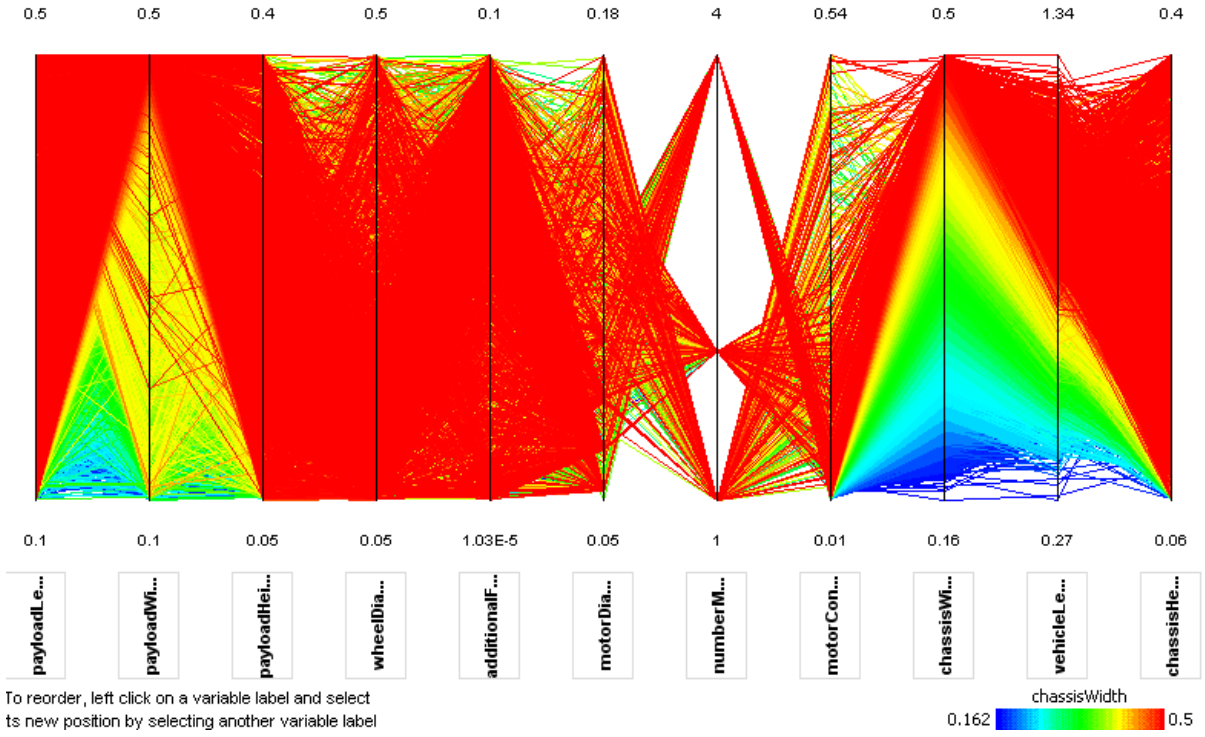

Figure 73: Parallel Coordinates of Chassis Design Variables Colored on Chassis Width

5.6 Chassis Structure Tradeoffs

The chassis structure is sized to support the mass of the structure as well as all of the weight that it must support. The mass of the chassis is based on its dimensions. This correlation is shown in Figure 74. It can also be seen that vehicle length correlates fairly well with chassis width as discussed in Section 0. These dimensions also correlate with the maximum deflection that the chassis could display.

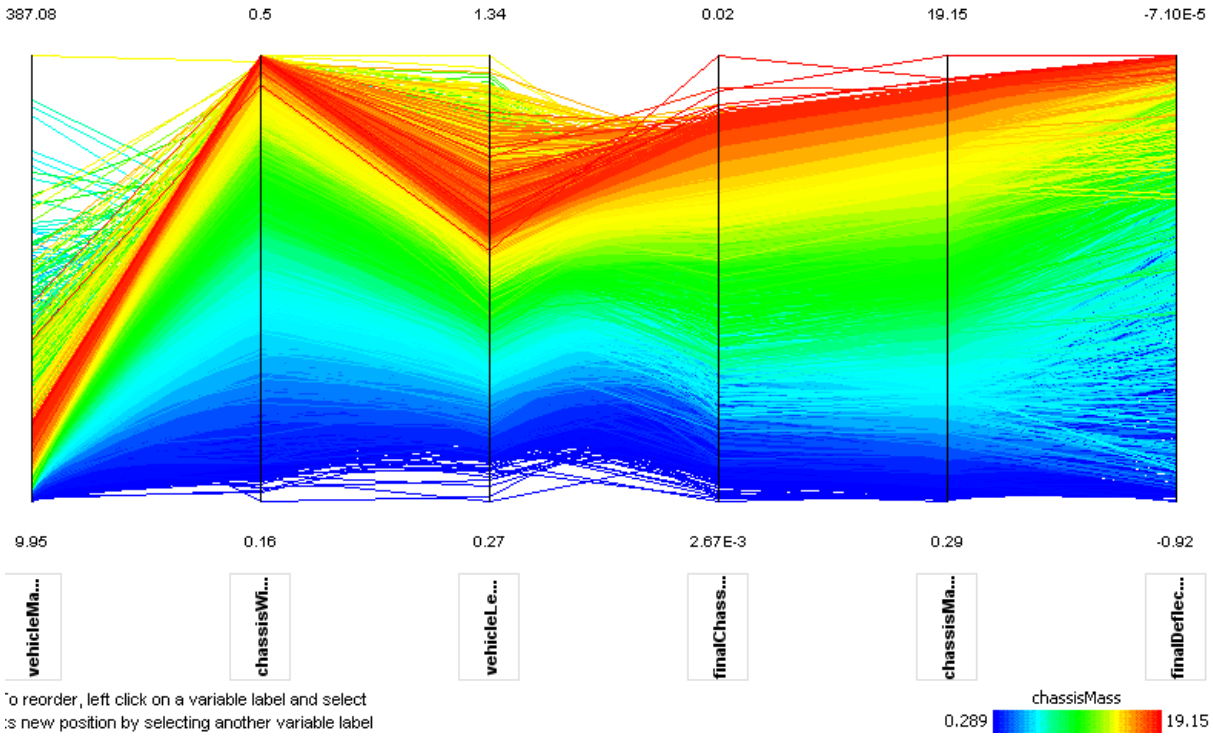

Figure 74: Parallel Coordinates of Chassis Design Variables Colored on Chassis Mass

5.7 Wheel/Track Tradeoffs

As given in Equation 52, wheel width is first sized based on a correlation between wheel diameter to wheel width and then increased if necessary to maintain a minimum ground pressure. When the wheeled vehicles that have had their wheel width increased (to maintain ground pressure) are brushed out, as shown in Figure 75, the total design count drops from 15,211 to 14,445. It can be seen that all of the wheels with excessively wide wheels (7^{th} axis) are removed. Removing all of the excessively wide wheels also removes all of the excessively heavy wheels.

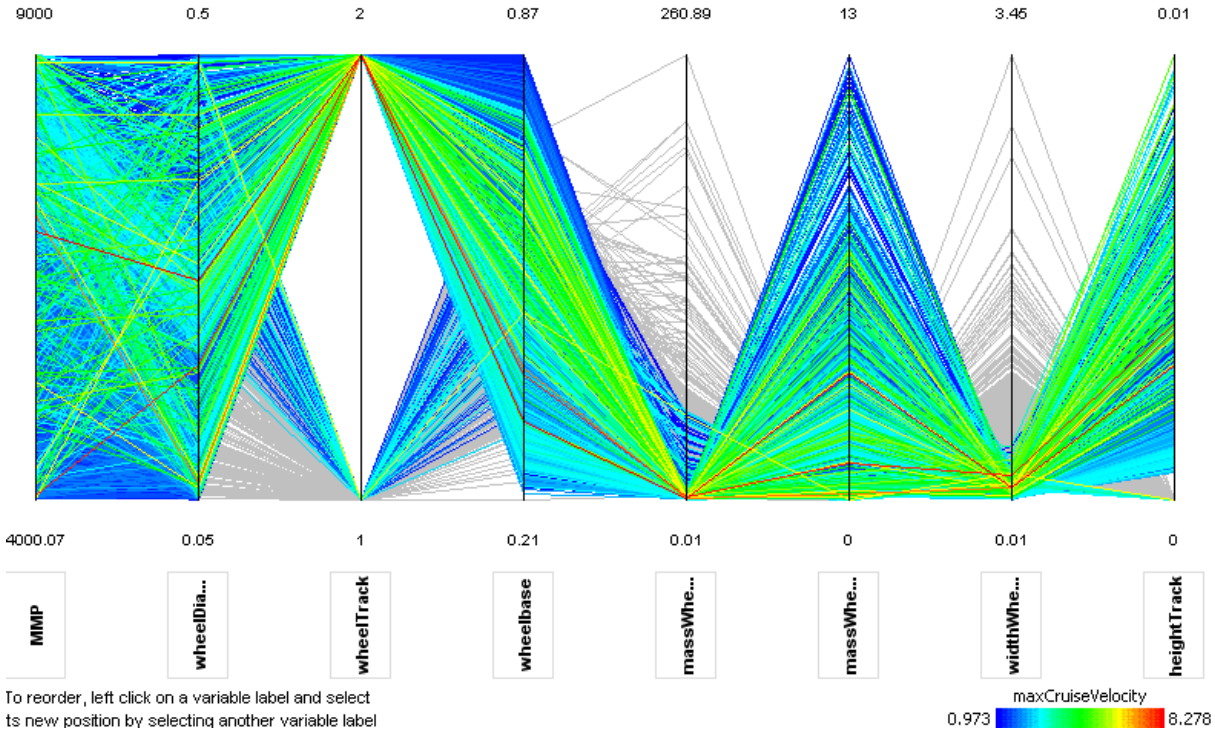

Figure 75: Parallel Coordinates of Wheel/Track Design Variables Colored on Velocity

The effect of wheel width being increased can be further examined with Figure 76. In addition to the wheeled vehicles, whose width had to be adjusted to achieve an equivalent ground pressure, tracked vehicles are brushed out. There are a few exceptions, but the vehicles with higher ground pressures have narrower wheels.

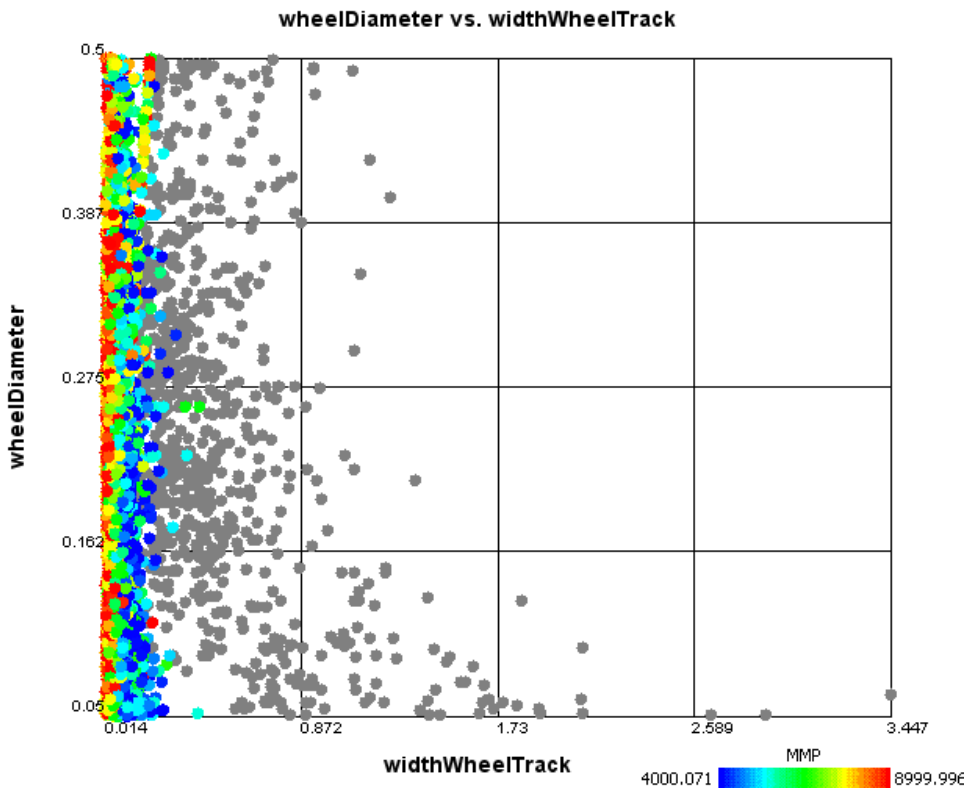

Figure 76: Wheel Diameter versus Wheel Width Colored based on Ground Pressure

5.8 Power Requirements Tradeoffs

Throughout this chapter it has been mentioned that maximum velocity, vehicle mass, and the mass allocated to the drive motors are very strongly correlated parameters. Maximum velocity and vehicle mass are plotted against one another in Figure 77 and are colored based on the total drive motor mass. In this two dimensional scatter plot it can again be seen that expending more mass on the drive motor and batteries results in a faster vehicle and also a heavier vehicle.

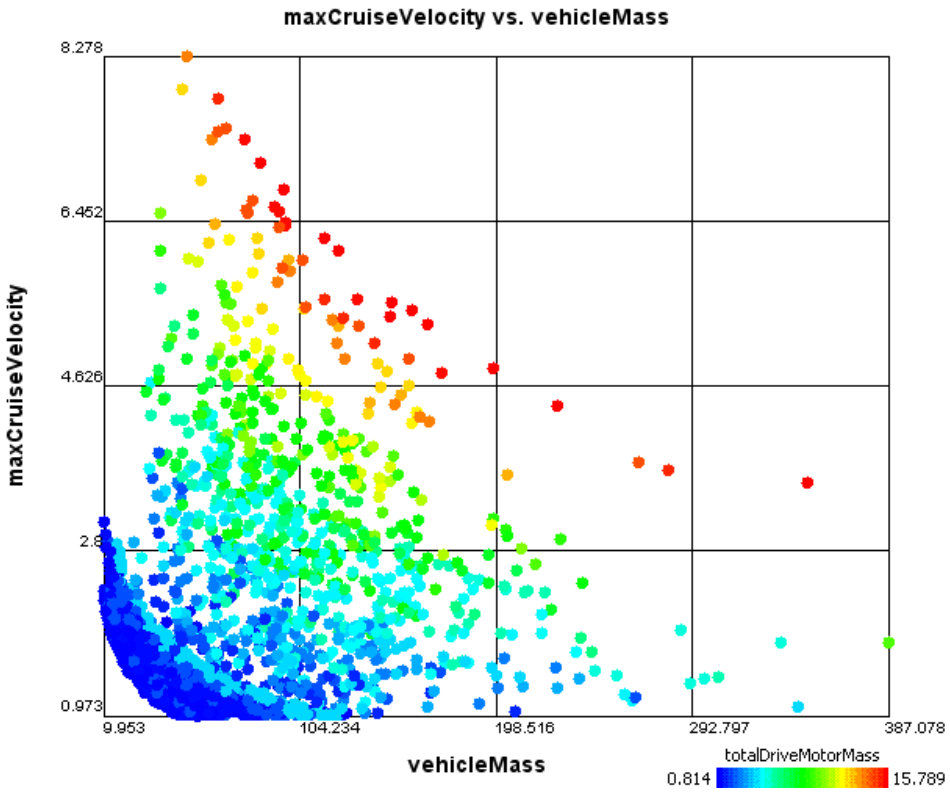

Figure 77: Maximum Velocity versus Vehicle Mass Colored On Motor Mass

5.9 Endurance Tradeoffs

Using the parallel coordinates plot shown in Figure 78, battery capacity is found to be very closely related to the two primary outputs from the Endurance subsystem: battery time and battery distance. As would be expected, implementation of a battery with a greater capacity for a robot of the same size results in a vehicle that can drive for longer periods of time. An increased distance that the vehicle can drive is to be expected because the calculation for battery distance is based on battery time as discussed in Section 3.10.2. Specifying a very large battery capacity on a very heavy robot will not necessarily achieve high endurance parameters. As discussed in Section 5.3, even allocating more mass to the batteries results in lower endurance.

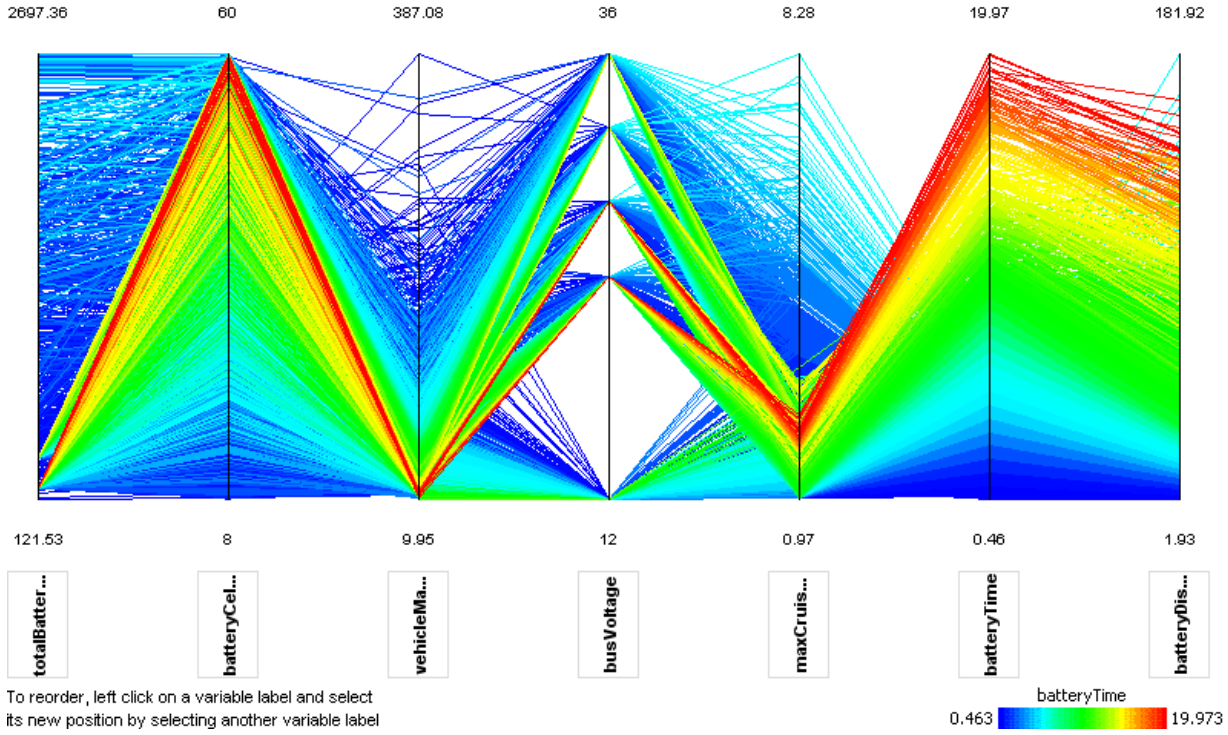

Figure 78: Parallel Coordinates of Endurance Variables Colored on Battery Time

5.10 Functional Capabilities Tradeoffs

It is important to study the design tradeoffs of the variables input to and output from the Functional Capabilities subsystem because these variables ultimately determine the usefulness of the robot. Maximum slope and traversal angles are the first parameters to be calculated in the Functional Capabilities subsystem; therefore, their tradeoffs are studied first. Figure 79 shows the failure mode (1st axis) corresponding to the maximum climbable slope (2nd axis), the total vehicle mass (3rd axis) and the failure mode (4th axis) corresponding to the maximum traversable slope (5th axis).

The limiting factor for slope climbing can never be greater than 36.44 [degrees] because the failure mode (mode #3) for friction is purely a function of the coefficient of friction which is assumed to be a constant value. As shown in Figure 79, these robots have large amounts of

torque. Very few robots fail due to tipping (failure mode #1). The rest of the robots fail due to having an insufficient amount of torque (failure mode #2).

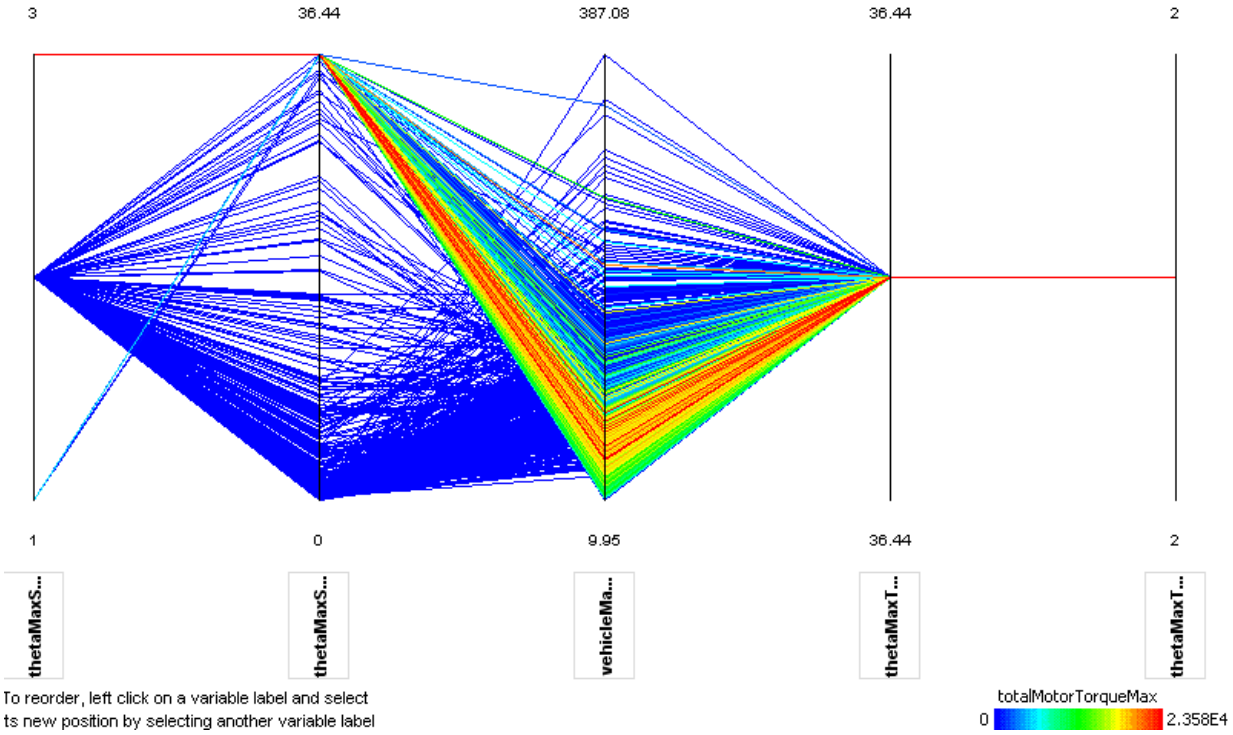

Figure 79: Parallel Coordinates of Slope Maneuvering Variables Colored on Motor Torque

Curb climbing also presents numerous tradeoffs. First, as shown in Figure 80, the minimum horizontal length (6^{th} axis) of ground required by the robot in order to climb over the curb of height specified in the 5^{th} axis, increases with the distance from the center of gravity to the rear axle of the vehicle (specified by color). Additionally, the vehicles requiring the largest platform length tend to most often have a failure mode associated with torque.

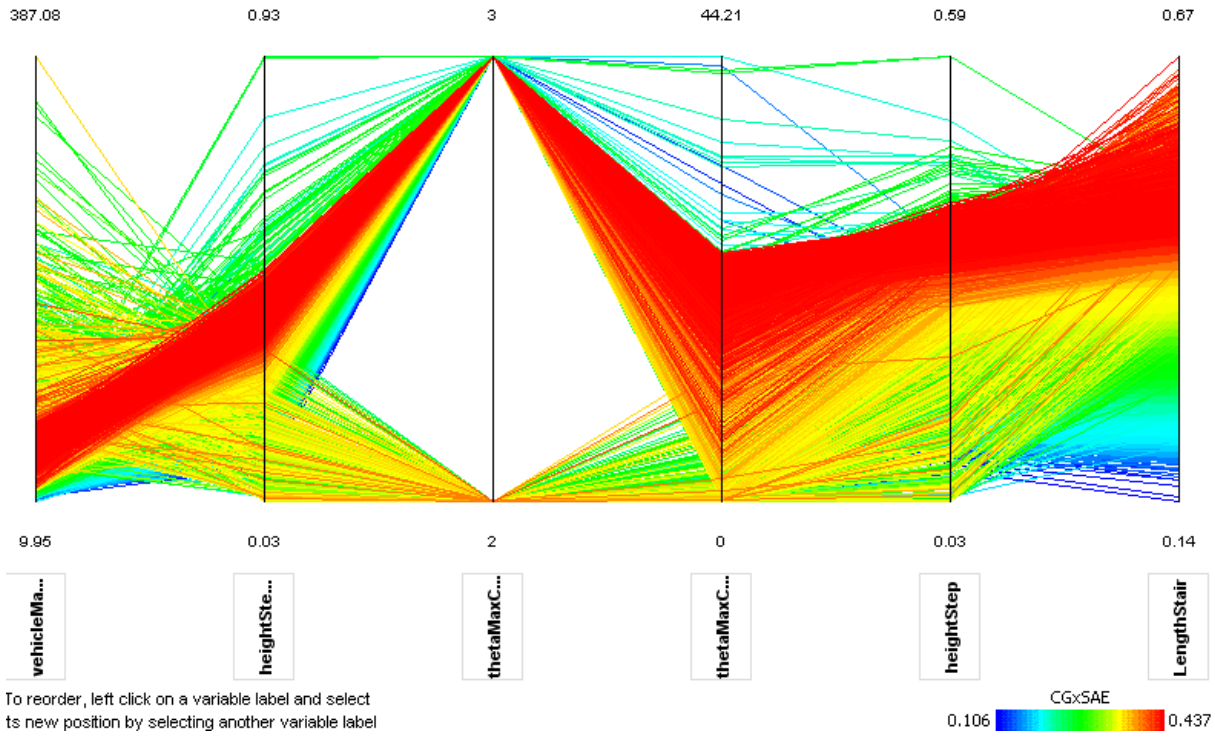

Figure 80: Parallel Coordinates of Curb Climbing Variables Colored on CGx

The height of a curb is plotted against the minimum platform length and colored based on the maximum curb climbing angle in Figure 81. Figure 82 shows the same two parameters plotted against one another; however, it is colored based on the failure mode that limits the variables on either axis. Here it can be seen that the robots colored blue fail in friction. Thus, in Figure 81, the maximum curb climbing angles of these robots colored blue are all the same, and the height of the curbs is a function of robot geometry.

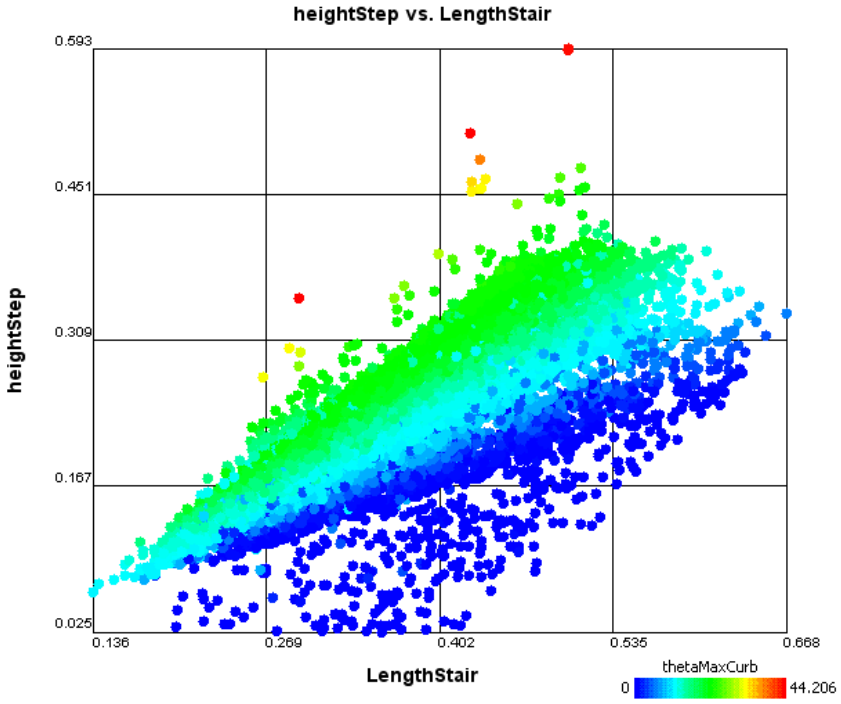

Figure 81: Curb Height versus Stair Length Colored on Maximum Curb Angle

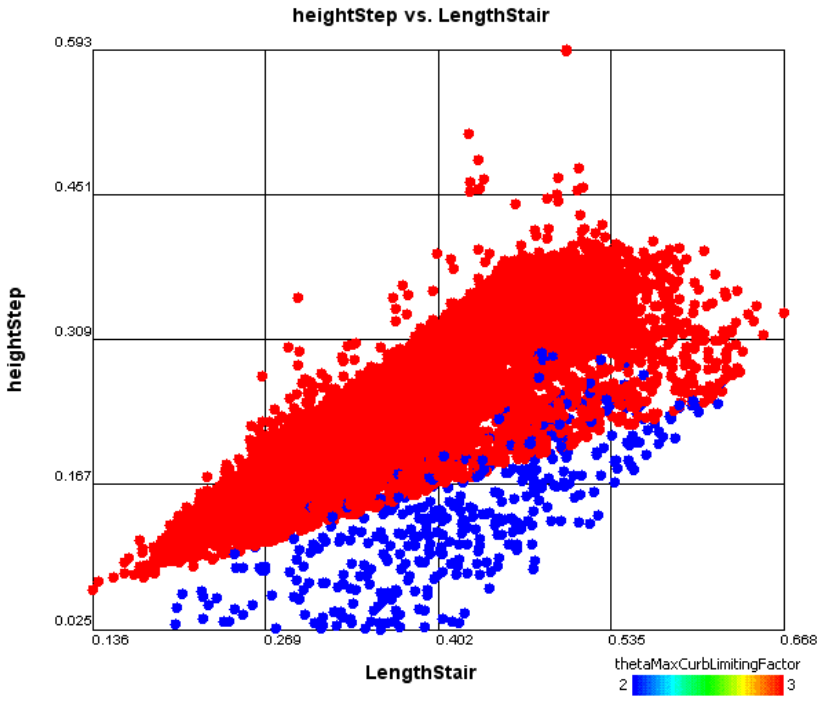

Figure 82: Curb Height versus Stair Length Colored on Curb Angle Limiting Factor

The next functional capability to be analyzed is the ditch crossing capability. It can be seen in Figure 83 that, as expected, the maximum crossable ditch width for a wheeled vehicle is purely a function of wheel diameter. This is determined by noticing the prismatic bands of color entering the wheel/track axis (2nd axis). Figure 83 also shows that the maximum crossable ditch width for tracks is directly correlated with the robot's wheelbase and distance from the center of gravity to the rear axle. Both of these variables are functions of the robot's length and wheel diameter.

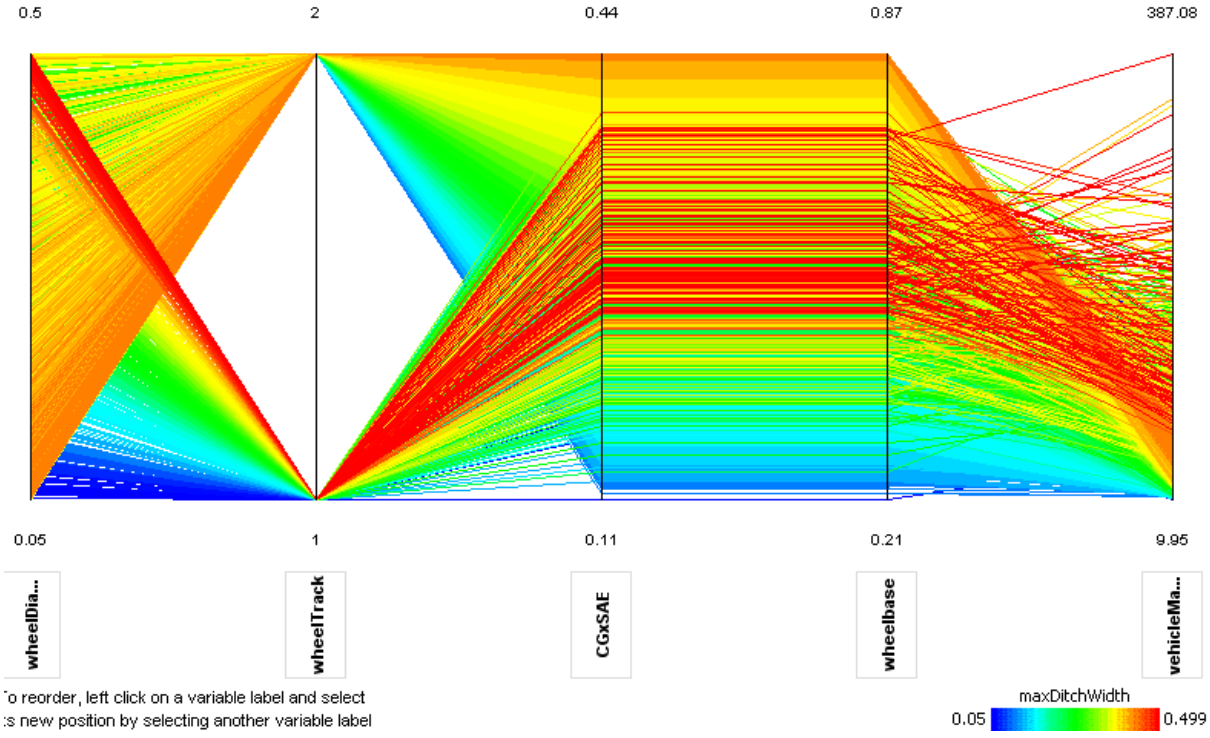

Figure 83: Parallel Coordinates of Ditch Crossing Variables Colored on Ditch Width

The ability of the robot to perform a zero radius turn (skid steer) is analyzed next. Robots that are not able to do a zero radius turn have been brushed out of the parallel coordinates plot shown in Figure 84. Of the 15,211 robots in the tradespace, only 680 (4.5%) do not have enough

torque to perform a zero radius turn on asphalt. Nonetheless, the minimum torque required by the motors in order to skid steer follow somewhat of a trend correlating with all of the variables used in the calculation.

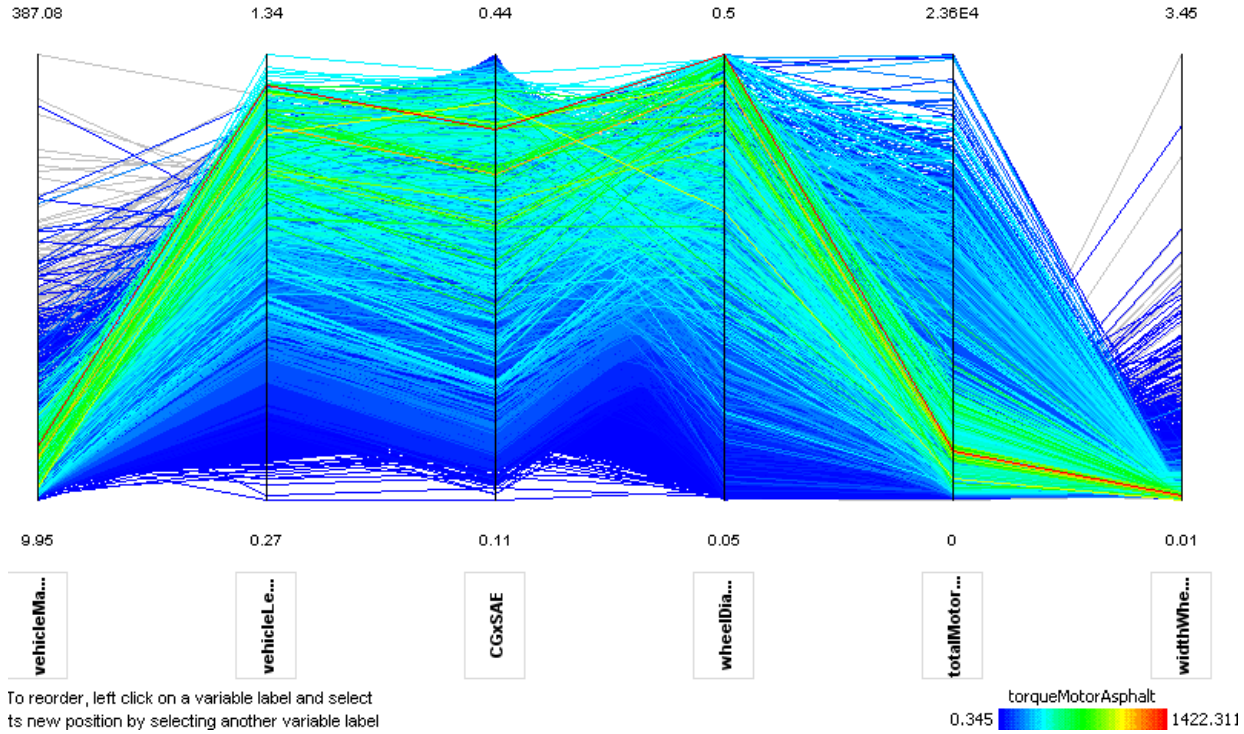

Figure 84: Parallel Coordinates of Skid Steer Variables Colored on Required Torque

The minimum hallway width through which the vehicle can maneuver exhibits dependencies on the distance from the center of gravity to the rear axle, wheel diameter, and vehicle length. Though the distance from the ground to the center of gravity of the robot is an input to the minimum hallway maneuverability equation, it does not seem to show evidence of driving the output.

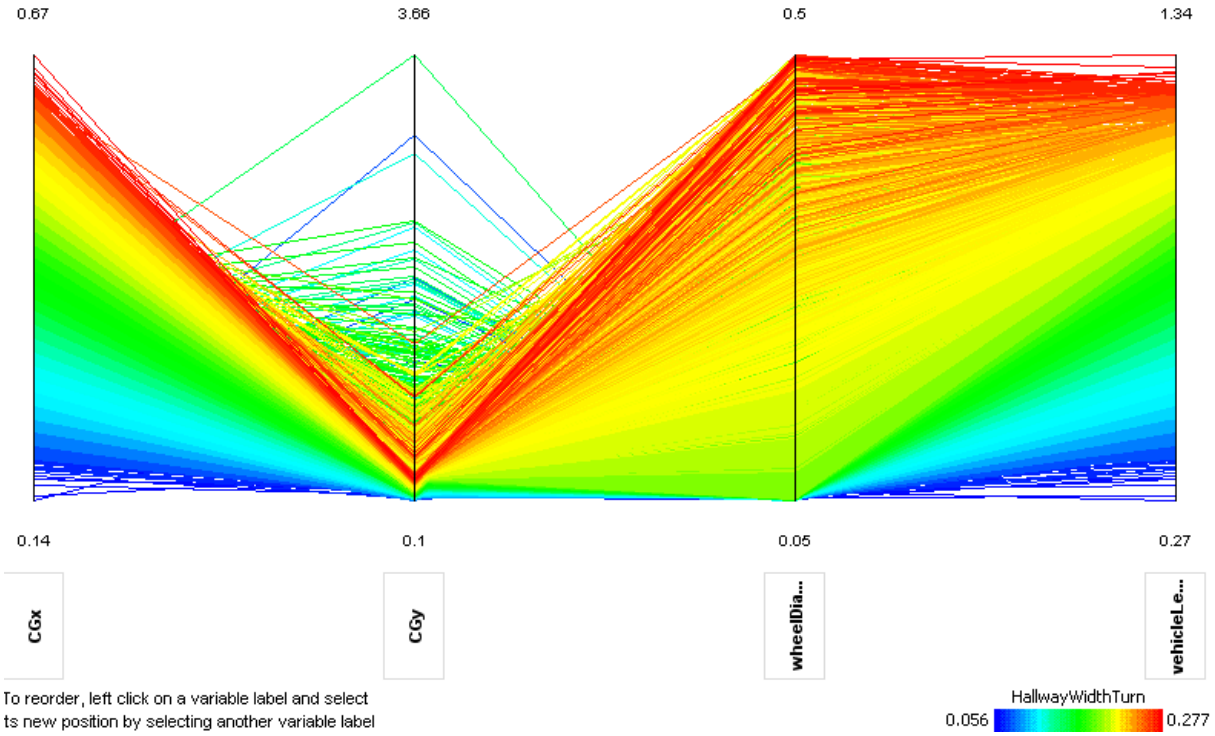

Figure 85: Parallel Coordinates of Hallway Width Variables Colored on Minimum Width

The maximum mass that a robot can drag has two failure modes: (1) the robot cannot apply enough torque to drag the mass and (2) the robot's wheels or tracks slip as it tries to drag the mass. These are referred to as the torque-limited and friction-limited cases, respectively. The effect of having two failure modes can be seen in Figure 86. The only parameter that varies in the friction limited case is vehicle mass. Total motor torque and wheel diameter are inputs to the torque limited case. In any regard, two separate bands of color can be seen in Figure 86.

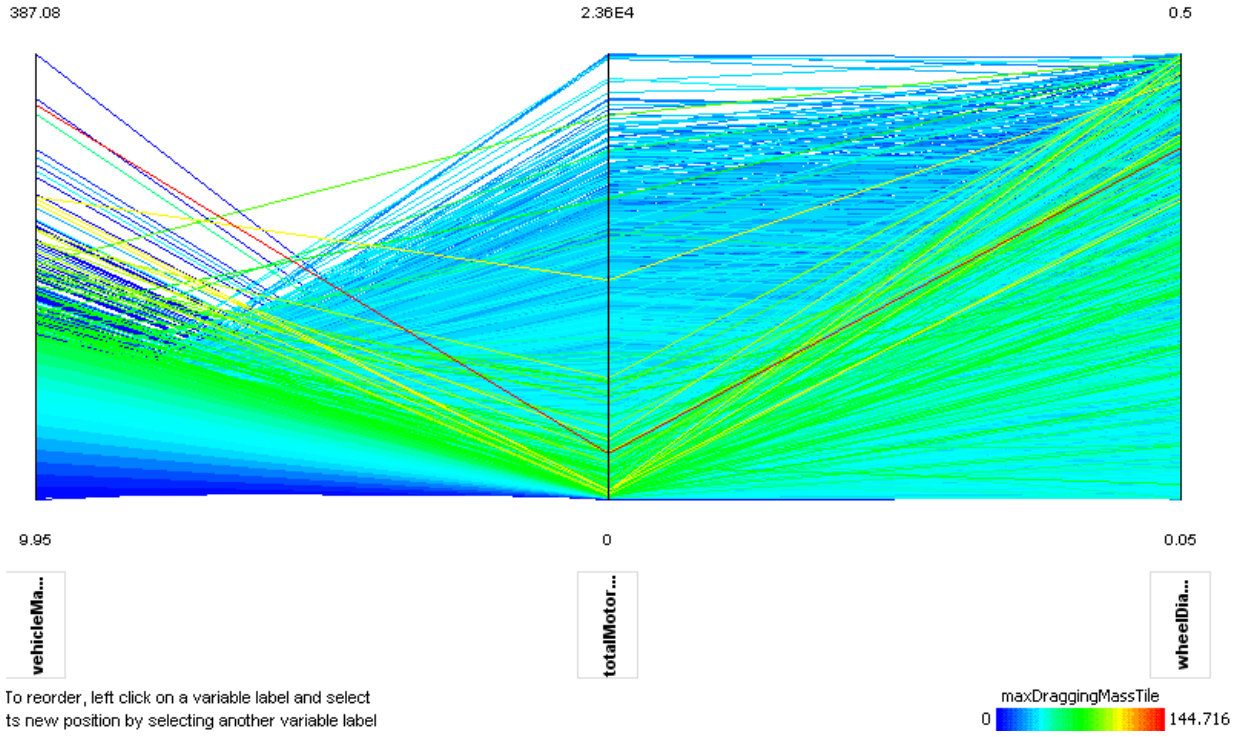

Figure 86: Parallel Coordinates of Dragging Force Variables Colored on Mass on Tile

Trade studies were presented in this chapter using the 15,211 feasible results generated from the model presented in Chapter 3. These trade studies show that several parameters work against one another (i.e., close lift capacity and far reach distance). These trade studies are important to understand when interpreting the results presented in the next chapter and when designing a robot.

Chapter 6

Robot Family GVI and PFPF Application

6.1 Product Family Motivation

The purpose of this work is to create a product family of robots for the location, identification, render-safe, recovery, and disposal of foreign or domestic objects. While the existing robot systems offered by companies like Foster-Miller (e.g., the Talon) and iRobot (e.g., the Packbot) are effective, there is no sharing or part commonality across their systems. As a result, users must maintain multiple sets of spare parts, tools, and repair manuals; keep multiple specialized technicians on staff for logistical and maintenance support; and conduct different sets of operator training and certification procedures for each robot since the operating systems and user controls are different for each robot. Furthermore, there is no plug-and-play capability across robot systems from different vendors since the operating systems and user controls are different for each robot. Furthermore, there is no plug-and-play capability across robot systems from different manufacturers, e.g., a manipulator arm from one manufacturer may not work on the other manufacturer's robot and vice versa. In this chapter, opportunities for commonality (and modularity) within the next-generation family of robots are investigated.

6.2 GVI Application

Starting with a set of requirements (e.g., range, lift capacity, weight, reach) for various robot missions, the goal is to determine a suitable level of commonality for the corresponding family of small, medium, and large robots that the military would like to field. We began by dissecting and analyzing several existing systems, including the Talon, Packbot, Bombot, and

RONS (see Figure 87). These systems were used to construct a "generic" robot architecture that was used for GVI analysis, which requires a reference design to study the impact of changes in one component or module on another.

(a) Bombot
http://www.defensetech.org

(b) Talon
http://www.foster-miller.com

(c) Packbot
http://www.irobot.com

(d) RONS
http://www.globalsecurity.org

Figure 87: Robots Dissected and Analyzed

The "generic" robot architecture is shown in Table 29 using a Design Structure Matrix (DSM). This DSM shows not only which components are connected to which, but also to what extent a change in one component likely impacts another component (L = low, M = medium, H = high) by taking into consideration change propagation throughout the system [30]. As seen in

Table 29, the DSM handles product variants, such as the individual components and connections varying by robot, by showing which subsystems are connected rather than individual components. Using this as the "generic" reference architecture for the robot family, the steps outlined in Figure 1 are used to compute GVI for each major subsystem in the robot.

Table 29: "Generic" Reference Architecture for GVI Analysis

	Chassis	Battery	Battery Bays	Flipper	Main Track	Com Box	Elect Box	Arm	Mast	Head	Gripper/ Wrist	3 Camera	Payload Bay	Aiming laser	Antenna	OCU
Chassis		M	M	M	M			M	L			L	H		L	
Battery	M						M									
Battery Bays	M															
Flipper	M				M											
Main Track	M			M												
Com Box							M					L			M	
Elect Box		M				M		L			L			L		
Arm	M						L		L	M	H	L				
Mast	L							L				L				
Head								M				L		L		
Gripper/Wrist							L	H								
3 Cameras	L					L		L	L	L				L		
Payload Bay	H															
Aiming Laser							L			L		L				
Antenna	L					M										M
OCU															M	

Table 30 shows the GVI matrix and resulting GVI values for the "generic" robot architecture. The matrix is read: "How much would meeting the requirement in (column name) cause redesign of the subsystem in (row name)?" Values of 1 = minimal redesign, 3 = some redesign, 6 = major redesign, 9 = complete redesign are used following GVI's redesign scale [5]. The scores in each row are then summed to achieve the GVI score for each subsystem. These scores are shown in the far right column of the table. This analysis was conducted by a group of graduate students in mechanical engineering and industrial engineering at Penn State University in conjunction with researchers and staff at the Applied Research Laboratory (ARL), several of whom had previous experience designing, building, operating, and testing robot systems [31]. Information was obtained through dissection of the existing robot systems as well as consultation

of each company's website and each robot's accompanying documentation and user manuals. Finally, the analysis results were discussed with several expert users who had experience operating these systems in the field.

Table 30: GVI Analysis for the "Generic" Robot Architecture

	Range (feet)	Slope Climb (deg)	Maneuver width (in)	On board vol (in^3)	On board wt (lb)	Drag/Roll/ Push (lb)	Horiz reach (in)	Vert high reach (in)	Sensing type	Video vert high reach (in)	Large Obj Pickup (length)	Large Obj Pickup (width)	Large Obj Pickup (height)	Lift capac (lb)	Tool precision	Tool size (in^3)	Tool wt (lb)	Comm range (ft)	GVI
chassis		3	3	6	6	6	6	1		1				6					38
battery					3	3													6
tracks		3	3			6													12
communication box	6														1			6	13
electronics box																			0
arm		1				6	9	9		9	3	3	3	9	3	3	3		61
gripper						6	1	1		1	6	6	6	3	6	6	6		48
cameras															1				1
payload bay				6	6				3										15
antennae	6																	6	12
OCU															1			3	4

To understand the results of the GVI analysis, chassis design is examined as an example. As seen in Table 30, it can be seen that the chassis' GVI score is high for many of the mission requirements listed along the top row, meaning that many different requirements drive the design of the chassis in different ways. Maneuvering as well as carrying capacity and reach requirements all impact chassis design, and each has a wide range of values across the missions provided for this study.

Such a high GVI score suggests that the chassis should vary in the robot family based on the needs of their missions; however, there may be opportunities to scale the chassis in one or more dimensions as discussed next. Unfortunately, this level of assessment is typically not addressed with GVI, and the implications of this are discussed in Sections 6.4 and 6.5.

A common length across all three robots would be very difficult to achieve unless some of the heavy-load-carrying scenarios are excluded. The length of the chassis has a direct impact on its ability to maneuver up stairs and over obstacles and also factors into the ability to turn into doors and hallways. Reaching or carrying objects also relies on a long, stable chassis and a lower center of mass to avoid tipping. The major tradeoffs for a common chassis length is lifting, stair climbing, and gap traversal versus maneuvering. The robot needs sufficient length to traverse stairs or ditches and to handle the heavy lifting requirements of the scenarios, but a robot that is too long could have limited maneuverability in hallways and openings. When performing trade studies, the goal is to find a suitable chassis length for picking up the heaviest objects at their respective arm extensions. Chassis lengths that satisfy this requirement and have the ability to maneuver through a standard sized doorway could be used as a common length.

Even if a common length is achieved, there might be adverse effects as many missions require little more than a small "scout" robot with no manipulator (e.g., the Bombot). Giving a robot a long chassis length would mean added weight and power consumption. It may also preclude it from fitting in small depressions or ditches for a closer look. Therefore, the maximum permissible value of any possible common length should be carefully evaluated. Sharing the length between just "scout" type and non-load-carrying robots would be easier to achieve and should be explored, since they have much higher maneuverability requirements than the load-carrying robots.

Common width across all robots would be very difficult to achieve; however, if the heavy-load carrying scenarios are excluded, then this again becomes easier to achieve. The non-object-manipulating scenarios may be prime candidates for a common width if two robots are needed. The robot's overall width includes the width of the chassis plus the width of the tracks

extending beyond the chassis edges. Narrow maneuvering requirements already imply a narrow robot including its tracks, while object carrying requirements generally imply wider widths. A robot attempting to drag very heavy weights over slightly uneven terrain would have to handle that weight shifting to either side of the robot, and widths narrower than a doorway might require a high weight and low height to prevent tipping. Exploration of the trade space can be used to confirm the feasibility of common chassis width for heavy-load-carrying robots. Meanwhile, sharing a common width across the non-heavy load carrying robots seems more feasible and should be examined further.

Finally, a common height for all robots is possible, especially across all non-heavy-load-carrying robots. The height dimension is basically limited by the minimum opening height requirement; any manipulator must also fit inside that same height. The missions that do not require manipulators could all share a common height. If an arm capable of dragging the weight from the heaviest dragging scenario can fold into a similar space (which may be possible), then all of the scenarios could have a common chassis height.

Similar analysis is applied to the other subsystems to arrive at the GVI matrix shown in Table 30. The complete analysis can be found in another work [31]. Based on the results in Table 30, several potential opportunities for commonality (see Table 31) are identified by looking at the low GVI values. Only the parameters that will be analytically evaluated with PFPF, in Section 6.3, are shown here. These opportunities include making the batteries, tracks, communication and electronic boxes, masts, cameras, payload bays, antennas, and OCUs common across the three robots. While some commonality is certainly possible, the chassis, arm, and gripper should not be part of the platform, but standardizing their interfaces facilitates plug-and-play capability in the family. GVI analysis could be complemented with a market

segmentation grid in future work; however, at this point segmentation is done solely on the basis of weight class.

Table 31: Possible Commonality Opportunities based on GVI Analysis

	Chassis			Mobility			Batteries			Manipulator		
Robot	Vehicle Length [m]	Chassis Width [m]	Chassis Height [m]	Wheels(=1)/ Tracks(=2)	Wheel Diameter [m]	Wheel or Track Width [m]	Battery Length [m]	Battery Width [m]	Battery Mass [kg]	Outer Arm Radius [m]	Arm Segment Length [m]	Number of Arm Links
Small												
Medium												
Large												

GVI Suggests Commonality

6.3 Product Family Optimization using PFPF

In parallel to the dissection and GVI analysis, a set of design rules were created for the robots to allow for the generation and analysis of new robot design alternatives as discussed in Chapter 3. Once these rules had been validated against the four existing robot systems (i.e., the Bombot, Talon, Packbot, and RONS) as shown in Chapter 4, a more analytical approach was taken to identify the platform in the robot family, namely, product family optimization using PFPF. The sampling and visual steering commands in ARL's Trade Space Visualization (ATSV) tool [32] were used to generate 15,211 robot design alternatives spanning the small, medium, and large robot design space. Each robot was then evaluated against its corresponding mission requirements (e.g., small robots were compared against the requirements for the missions that needed small robots). The effectiveness of each robot is computed based on the

corresponding set of requirements by comparing the calculated value of the requirement, R_i, to the Threshold, T_i, and Objective, O_i, values for that requirement as follows:

1. $Eff_{threshold}$, Threshold Effectiveness (i.e., how well are the threshold requirements met?):

For requirements with min. values: *For requirements with max values:*

$$Eff_{threshold} = \frac{\sum eff_{ti}}{\sum w_i} \quad where: \begin{Bmatrix} eff_{ti} = \frac{R_i}{T_i} w_i & if & R_i < T_i \\ eff_{ti} = 1 \times w_i & if & R_i \geq T_i \end{Bmatrix} \; or \; \begin{Bmatrix} eff_{ti} = \frac{T_i}{R_i} w_i & if & R_i > T_i \\ eff_{ti} = 1 \times w_i & if & R_i \leq T_i \end{Bmatrix} \quad (78)$$

2. $Eff_{objective}$, Objective Effectiveness (i.e., how well are the objective requirements met?):

For requirements with min. values: *For requirements with max values:*

$$Eff_{objective} = \frac{\sum eff_{oi}}{\sum w_i} \quad where: \begin{Bmatrix} eff_{oi} = \frac{R_i}{O_i} w_i & if & R_i < O_i \\ eff_{oi} = 1 \times w_i & if & R_i \geq O_i \end{Bmatrix} \; or \; \begin{Bmatrix} ef_{oi} = \frac{O_i}{R_i} w_i & if & R_i > O_i \\ eff_{oi} = 1 \times w_i & if & R_i \leq O_i \end{Bmatrix} \quad (79)$$

3. $Eff_{overall}$, Overall Effectiveness (i.e., how well are the requirements met overall?):

For requirements with min. values: *For requirements with max values:*

$$Eff_{overall} = \frac{\sum eff_i}{\sum w_i} \quad where: \begin{Bmatrix} eff_i = 0 & if & R_i < T_i \\ eff_i = \frac{R_i}{O_i} w_i & if & T_i \leq R_i < O_i \\ eff_i = 1 \times w_i & if & R_i \geq O_i \end{Bmatrix} \; or \; \begin{Bmatrix} eff_i = 0 & if & R_i > T_i \\ eff_i = \frac{O_i}{R_i} w_i & if & T_i \geq R_i > O_i \\ eff_i = 1 \times w_i & if & R_i \leq O_i \end{Bmatrix} \quad (80)$$

Effectiveness is thus computed as a percentage of the requirements that were met. As calculated using Equation (78 through Equation (80, an effectiveness of 100% means that the robot meets all of the requirements; 0% means that none of the requirements are met.

Upon completion of the effectiveness calculations and upon being classified as small, medium or large robots, each size was sorted by highest to lowest average effectiveness. The top performing 90 robots of each size were considered candidates for the product family as shown in Figure 88. This filtering process reduced the number of robot designs to 270 promising candidates: 90 designs for the small robot, 90 for the medium robot, and 90 for the large robot. Even though this is a relatively small number of robot designs, it leads to 729,000 (= 90 x 90 x 90) possible robot families when every possible combination of one small, one medium, and one large robot is considered. In preliminary studies a minimum value for effectiveness was used as a cutoff. However, generation of a larger number of robots resulted in numerous robots with very similar effectiveness values. Additionally, the small and medium robots have significantly higher effectiveness values than the large robots. Therefore, it was determined that the largest number of families would be evaluated as possible. Additionally, an equal number of small, medium, and large robots would be considered. The most effective 90 robots in each size were selected because addition of more robots would run the computer out of memory. A computer program was written in Java™ to create every combination of small, medium, and large robots. This code also calculates PFPF and average effectiveness for each family.

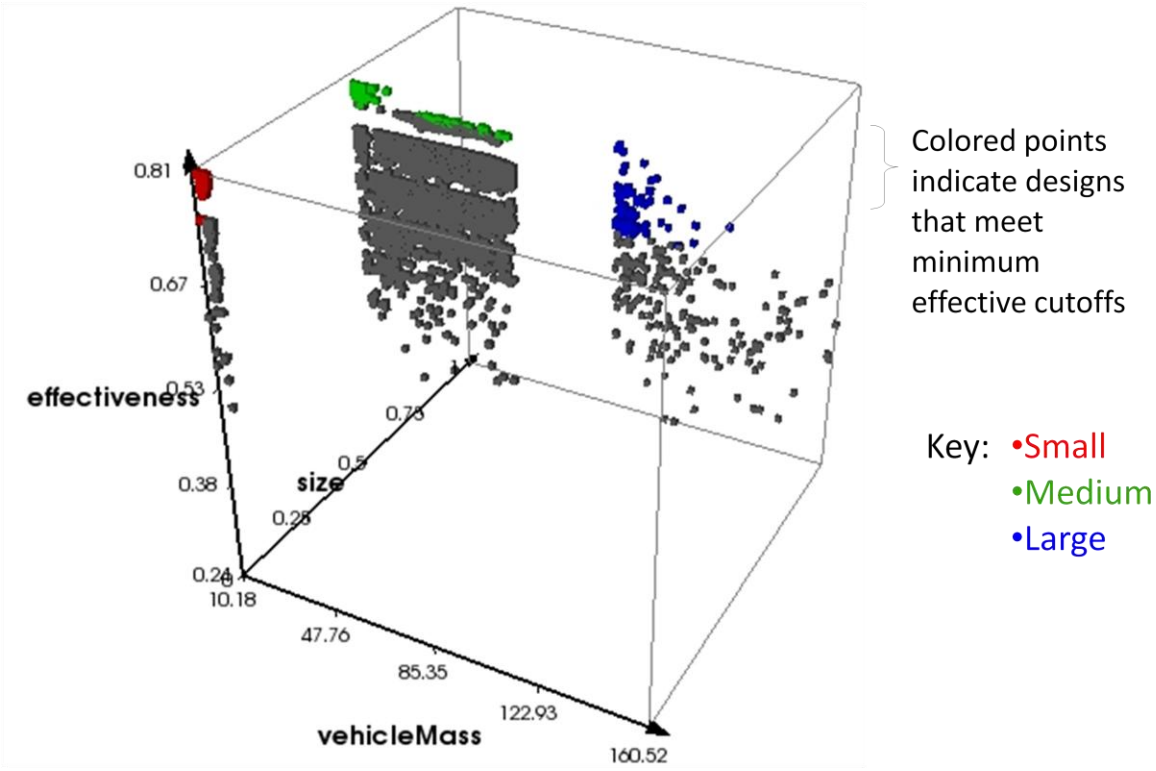

Figure 88: Effectiveness vs. Robot Size & Mass

A sensitivity analysis was performed to determine which objective requirements are the most difficult to meet. The small, medium and large robots consistently have difficulty meeting the slope climbing, staircase climbing, far reach distance, and endurance requirements. The medium and large robots consistently have difficulty meeting the slope traversal, payload mass and lift capacity. The large robots additionally struggle to meet stair climbing requirements.

Even though visual steering tools were used within ATSV during trade space exploration, it can be seen in Figure 88 that a significantly higher quantity of medium robots were created than small or large robots. This is because the genetic algorithm within ATSV depends on a "seed" for which to create more similar designs. Thus, if most of the designs are medium or large robots, the genetic algorithm will have difficulty determining which input parameters will

result in a small robot. Even though the model may be visually steered toward creation of a small robot, it does not guarantee a small robot will be output from the model.

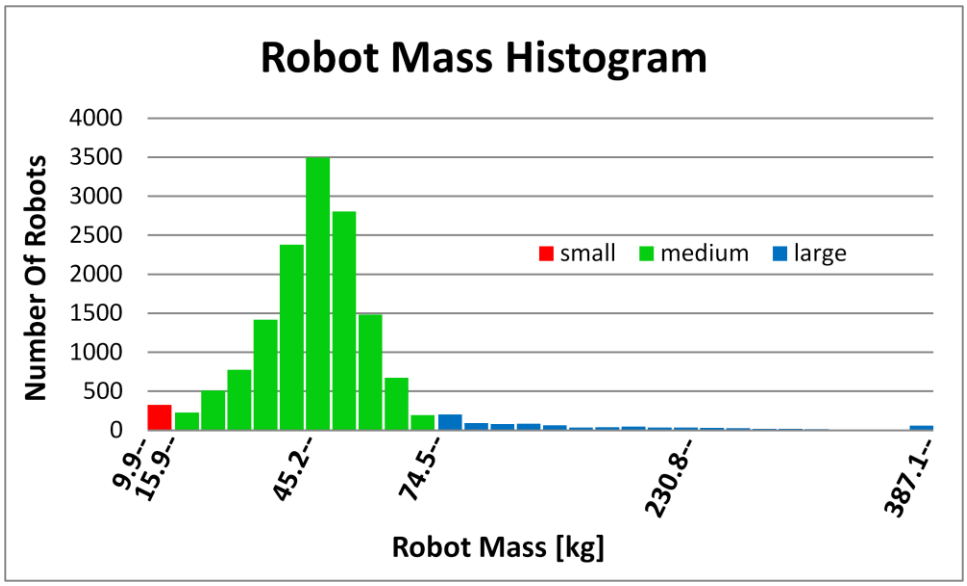

Figure 89: Robot Mass Histogram

For each one of these 729,000 possible robot families, two calculations are made. First PFPF is calculated as shown in Equation 1. The parameters included in the PFPF analysis include vehicle length, chassis width, chassis height, ground clearance, wheel or track configuration, wheel diameter, wheel/track width, battery length, battery width, battery mass, number of batteries, drive motor diameter, drive motor length, manipulator segment radius, manipulator segment length, and the number of manipulator segments. Secondly, the objective overall effectiveness is calculated. The objective overall effectiveness, which is the average effectiveness across the three robots, is defined in Equation 81.

$$\text{Obj\_Eff\_Overall} = \frac{Eff_{overall,Small} + Eff_{overall,Medium} + Eff_{overall,Large}}{3} \quad (81)$$

Table 32 shows examples of the robot families enumerated in this study. Each row indicates a family of robots where the number under the small, medium, and large column headings refer to the row index of the robots in the data set that remain after filtering out the ineffective robot designs (i.e., not the most effective 90 for each size).

Table 32: Example of Robot Family Analysis

Family Number	Small Robot	Medium Robot	Large Robot	Dissimilarity (PFPF)	Overall Effectiveness ($Eff_{overall}$)			Avg_Eff_ Overall
					Small	Medium	Large	
1	1	1	1	7.342	89.182%	88.855%	79.571%	85.869%
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
3881	1	44	11	8.552	89.182%	89.865%	81.079%	86.709%
3882	1	44	12	8.135	89.182%	89.865%	81.389%	86.812%
3883	1	44	13	7.420	89.182%	89.865%	79.977%	86.341%
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
5482	1	61	81	6.956	89.182%	89.545%	79.988%	86.238%
5483	1	61	82	4.577	89.182%	89.545%	79.313%	86.013%
5484	1	61	83	6.345	89.182%	89.545%	78.518%	85.748%
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
248681	31	62	11	6.084	89.404%	89.563%	81.079%	86.626%
248682	31	62	12	5.820	89.404%	89.563%	81.389%	86.730%
248683	31	62	13	6.294	89.404%	89.563%	79.977%	86.259%
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
729000	90	90	90	8.297	82.924%	87.333%	79.538%	81.646%

Using the data in Table 32, Obj_Eff_Overall is plotted versus PFPF to identify promising robot families. The result is shown in Figure 90, which shows the tradeoff between commonality (i.e., low PFPF values) and performance (i.e., high effectiveness) within the robot families. Preliminary trade studies involving fewer product family options showed a much more dramatic tradeoff between commonality and performance; however, due to more densely populating the tradespace with design alternatives, numerous good design choices can exist. In any regard, in Figure 90, each point represents a robot family, and the most promising robot families are marked with +'s. Color indicates preference in the figure: red points are more preferred to blue points, assuming an equal importance weighting is placed on PFPF and Effectiveness. The six families marked by the +'s represent the Pareto frontier among the designs in that they dominate all of the other families by offering the best combination of commonality and overall objective effectiveness.

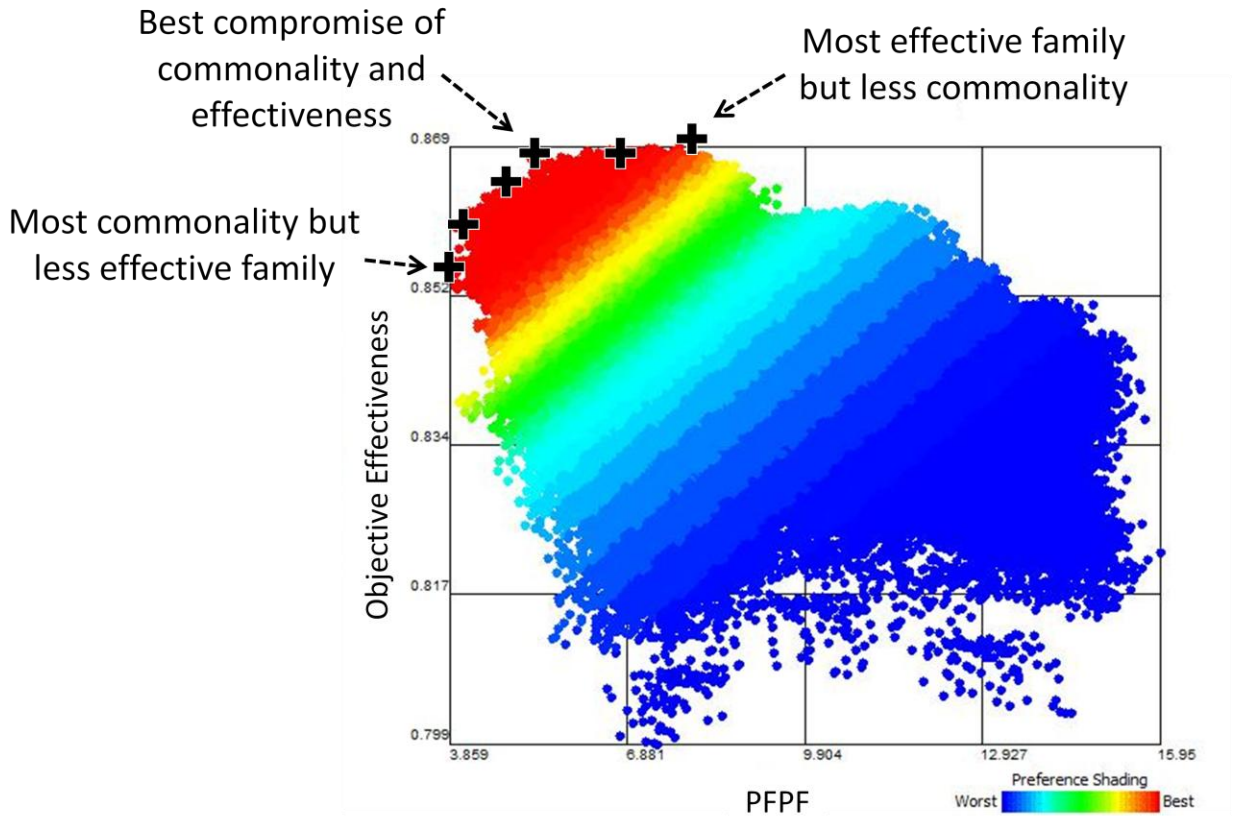

Figure 90: PFPF vs. Objective Effectiveness

Of these six families, three are of particular interest as highlighted in the figure: (1) the Most Effective Family, (2) the Most Common Family, and (3) the Best Compromise Family. The Most Effective Family does the best job of all the families in satisfying the effectiveness requirements for the small, medium and large robots (Objective Effectiveness = 86.8%), but it has less commonality (higher PFPF) than the other designs, although it is by no means the worst among those evaluated. The Most Common Family is the reverse – it has the most commonality among the three designs, but this comes at a slight loss in performance (Objective Effectiveness = 86.0%). Finally, the Best Compromise Family falls between the two – it has more commonality than the Most Effective Family but with less sacrifice in performance compared to

the Most Common Family. In fact, its Objective Effectiveness = 86.7%, indicating a remarkably good compromise in this robot family.

The corresponding parameter settings for these three robot families are listed in Table 33. Here, color is used to highlight parameter values that are common (in orange) and similar (in yellow), i.e., within 5% across two or more robots within a given family. Note that even though some of the parameter values are the same across families (i.e., they all use tracks, and nearly all of them have the same battery specifications), the color coding for common and similar parameter values are within a single robot family, not across the three robot families. For example, it is only a coincidence that the Large Robot in the Best Compromise Family has a similar value to the Large Robot in the Most Common Family for vehicle length.

Table 33: Possible Common, Similar, and Unique Parameter Settings in the Robot Family

	Robot	Vehicle Length [m]	Chassis Width [m]	Chassis Height [m]	Ground Clearance [m]	Wheels(=1)/ Tracks(=2)	Wheel Diameter [m]	Wheel or Track Width [m]	Battery Length [m]	Battery Width [m]	Battery Mass [kg]	Drive Motor Diameter [m]	Drive Motor Length [m]	Outer Arm Radius [m]	Arm Segment Length [m]	Number of Arm Links
Best Compromise Family	Small	0.542	0.206	0.198	0.019	2	0.264	0.033	0.112	0.062	1.4	0.064	0.096	0.021	0.418	3
	Medium	0.788	0.416	0.249	0.016	2	0.078	0.039	0.112	0.062	2.8	0.064	0.096	0.021	0.243	3
	Large	1.007	0.498	0.257	0.021	2	0.175	0.058	0.112	0.062	2.8	0.064	0.096	0.021	0.229	3
Most Common Family	Small	0.543	0.224	0.135	0.036	2	0.251	0.025	0.112	0.062	1.4	0.079	0.112	0.021	0.105	3
	Medium	0.732	0.409	0.163	0.017	2	0.051	0.047	0.112	0.062	1.4	0.063	0.096	0.021	0.283	3
	Large	1.007	0.475	0.170	0.030	2	0.178	0.049	0.112	0.062	1.4	0.052	0.083	0.021	0.218	3
Most Effective Family	Small	0.543	0.224	0.135	0.036	2	0.251	0.025	0.112	0.062	1.4	0.079	0.112	0.021	0.105	3
	Medium	0.792	0.418	0.236	0.010	2	0.057	0.033	0.112	0.062	1.4	0.051	0.082	0.021	0.292	3
	Large	0.763	0.371	0.117	0.010	2	0.115	0.108	0.112	0.062	2.8	0.064	0.096	0.022	0.408	3

= Common Values # = Similar (< 5%) Values

Based on the results in Table 33, there are several opportunities for using common and similar components within the robot family as revealed by PFPF optimization. As expected, the Most Effective Family, i.e., the one that is able to complete the mission scenarios most

effectively, has the least commonality among the specific parameter settings as seen from the color coding, and while it may appear that there is more similarity in the Best Compromise Family, there is less variability in the Most Common Family, which gives it a lower PFPF score (i.e., higher commonality). In terms of each family:

Best Compromise Family: There seems to be a much higher degree of commonality and similarity between the medium and large robots in this family. A common drive motor can be used across the three tracked vehicles. All three robots utilize a common BB2590U battery: the small robot uses one while the medium and large use two. The manipulator is able to share common segment radii across their three segments. The medium and large robots look like they can share chassis height and manipulators. The small and large robots may be able to share a common ground clearance.

Most Common Family: This family seems to be able to share the exact same battery specifications across these three tracked vehicles. All three robots are equipped with the same manipulator segment radii and three manipulator segments. The chassis height can again be shared between the medium and large robots. Additionally, the medium and large robots can share a common track width.

Most Effective Family: The Most Effective Family consists of three tracked vehicles. All three robots have three segment links of which segment radii is common. Like the Best Compromise Family, all three robots utilize common BB2590U batteries: the small and medium robots in Most Effective Family have two BB2590U batteries while the large has two. It is again noticed that commonality tends to favor similarity among the medium and large robots in a family. Similar parameters between the medium and large robots include vehicle length and ground clearance.

Finally, looking across the families, it appears that tracked vehicles are consistently the most effective. There also appear to be opportunities for scaling the quantity of BB2590U batteries used across the family. There appear to be several opportunities to share some manipulator parameters within a robot family.

6.4 GVI to PFPF Comparison

At this point, there are two different sets of suggestions for commonality based on GVI and PFPF analysis. The question to consider is: are the results the same, i.e., do GVI and PFPF recommend the same platform for the robot family? Does the perception-based approach (GVI), which is subjective to expert opinion, provide comparable results to the analytical approach (PFPF), which relies on detailed models? The following comparisons examine the four major subsystems – chassis, mobility, batteries, and manipulator – that were considered during both GVI and PFPF analysis to answer these questions.

Chassis: The GVI suggests that a large portion of the chassis design should be common. The PFPF suggests that very few of the chassis parameters be similar. In Table 34, the parameters for which the GVI and PFPF agree should be common are highlighted in green, those that the GVI and PFPF disagree on being common are highlighted in red, and the parameters that are not highlighted are not recommended by either the GVI or PFPF to be common. The GVI and PFPF suggestions show only partial agreement when considering the Best Compromise and Most Common families. The only commonality suggestions offered by the PFPF are for the chassis height in the medium and large robots of the Best Compromise and Most Common families. The GVI recommends that all three chassis heights be common. In disagreement with the PFPF, the GVI suggests that the small and medium robots share a common length and width.

The Most Effective family recommends the exact opposite vehicle lengths be held common to those suggested by the GVI (e.g., the GVI recommends common vehicle length for the small and medium, while PFPF suggests commonality of the medium and large).

Table 34: Comparison of Commonality Recommendations of GVI and PFPF

		Chassis			Mobility			Batteries			Manipulator		
	Robot	Vehicle Length [m]	Chassis Width [m]	Chassis Height [m]	Wheels(=1)/ Tracks(=2)	Wheel Diameter [m]	Wheel or Track Width [m]	Battery Length [m]	Battery Width [m]	Battery Mass [kg]	Outer Arm Radius [m]	Arm Segment Length [m]	Number of Arm Links
Best Compromise Family	Small	**0.542**	**0.206**	**0.198**	**2**	0.264	0.033	**0.112**	**0.062**	**1.4**	**0.021**	**0.418**	**3**
	Medium	**0.788**	**0.416**	**0.249**	**2**	0.078	0.039	**0.112**	**0.062**	**2.8**	**0.021**	**0.243**	**3**
	Large	1.007	0.498	**0.257**	**2**	0.175	0.058	**0.112**	**0.062**	**2.8**	**0.021**	**0.229**	**3**
Most Common Family	Small	**0.543**	**0.224**	**0.135**	**2**	0.251	0.025	**0.112**	**0.062**	**1.4**	**0.021**	**0.105**	**3**
	Medium	**0.732**	**0.409**	**0.163**	**2**	0.051	**0.047**	**0.112**	**0.062**	**1.4**	**0.021**	**0.283**	**3**
	Large	1.007	0.475	**0.170**	**2**	0.178	**0.049**	**0.112**	**0.062**	**1.4**	**0.021**	**0.218**	**3**
Most Effective Family	Small	**0.543**	**0.224**	**0.135**	**2**	0.251	0.025	**0.112**	**0.062**	**1.4**	**0.021**	**0.105**	**3**
	Medium	**0.792**	**0.418**	**0.236**	**2**	0.057	0.033	**0.112**	**0.062**	**1.4**	**0.021**	**0.292**	**3**
	Large	**0.763**	0.371	**0.117**	**2**	0.115	0.108	**0.112**	**0.062**	**2.8**	**0.022**	**0.408**	**3**

#	= GVI & PFPF Commonality Suggestions Agree
#	= GVI & PFPF Commonality Suggestions Disagree
#	= Neither Suggest Commonality

Mobility: Neither the GVI nor PFPF suggest much commonality among the mobility parameters (see Table 34). GVI suggests no commonality (with the exception of all tracked robots). For the Most Common Family, PFPF disagrees with the GVI recommendation by suggesting a common track width.

Batteries: GVI suggests using the same batteries, just increasing the number of these batteries used, to obtain the necessary power for the three robots. The Most Common Family exactly matches this suggestion as shown in Table 34.

Manipulator: Studying Table 34, PFPF suggests commonality for the outer radii of each segment as well as the number of segment links for all three robots. The GVI suggests commonality of only those parameters for the small and medium robots. PFPF suggests no commonality for arm segment length while the GVI suggests that the lengths should be common for the small and medium robots.

Ultimately, it can be seen that there are a significant number of overlapping suggestions for commonality when comparing the results of GVI and PFPF. This is a promising first step in validating the use of GVI for product family design, i.e., to have results match so well for a perception-based approach when compared to a rigorous analytical approach using optimization. It has been difficult to compare the results of GVI in published studies as few analytical models exist for the same family. In this robot family example, both options could be explored given the duration and nature of the project. Thus, the results of this analysis are offered as a unique contribution of this work. The question still remains, however: which commonality recommendations should be used – those from GVI or those from PFPF?

6.5 Combining GVI With PFPF for Optimization

The 729,000 families created from the robots output from the updated model were also compared to the GVI commonality recommendations. Unlike the preliminary study, the updated families contain five families that exhibit all of the GVI commonality recommendations. A few hundred more families lack only one common parameter from the GVI commonality recommendations. Table 35 shows the five robot families that exhibit all of the same common parameters as suggested by the GVI. In this table, the parameters that the GVI and PFPF both suggest to be common are highlighted in green. The parameters for which the PFPF suggests

more commonality than suggested by the GVI are highlighted in red. These parameters highlighted in red indicate that the GVI may have overlooked a feasible commonality suggestion. Finally, those parameters not highlighted are not recommended by either the GVI or PFPF to be common.

		Chassis			Mobility			Batteries			Manipulator		
	Robot	Vehicle Length [m]	Chassis Width [m]	Chassis Height [m]	Wheels(=1)/ Tracks(=2)	Wheel Diameter [m]	Wheel or Track Width [m]	Battery Length [m]	Battery Width [m]	Battery Mass [kg]	Outer Arm Radius [m]	Arm Segment Length [m]	Number of Arm Links
Family 1	Small	**0.557**	**0.227**	**0.318**	**2**	**0.261**	0.028	**0.112**	**0.062**	**1.4**	**0.021**	**0.565**	**3**
	Medium	**0.592**	**0.221**	**0.334**	**2**	**0.291**	0.032	**0.112**	**0.062**	**1.4**	**0.021**	**0.524**	**3**
	Large	0.665	0.301	**0.344**	**2**	0.181	0.13	**0.112**	**0.062**	**1.4**	**0.021**	0.306	**3**
Family 2	Small	**0.544**	**0.203**	**0.079**	**2**	**0.269**	0.034	**0.112**	**0.062**	**1.4**	**0.021**	**0.134**	**3**
	Medium	**0.575**	**0.191**	**0.086**	**2**	**0.279**	0.043	**0.112**	**0.062**	**1.4**	**0.021**	**0.133**	**3**
	Large	0.911	0.5	**0.079**	**2**	0.121	0.061	**0.112**	**0.062**	**1.4**	**0.021**	0.112	**3**
Family 3	Small	**0.578**	**0.208**	**0.08**	**2**	**0.277**	0.03	**0.112**	**0.062**	**1.4**	**0.021**	**0.569**	**3**
	Medium	**0.603**	**0.205**	**0.08**	**2**	**0.297**	0.035	**0.112**	**0.062**	**1.4**	**0.021**	**0.568**	**3**
	Large	0.911	0.5	**0.079**	**2**	0.121	0.061	**0.112**	**0.062**	**1.4**	**0.021**	0.112	**3**
Family 4	Small	**0.646**	**0.223**	**0.35**	**2**	**0.307**	0.025	**0.112**	**0.062**	**1.4**	**0.021**	**0.104**	**3**
	Medium	**0.608**	**0.224**	**0.32**	**2**	**0.301**	0.035	**0.112**	**0.062**	**1.4**	**0.021**	**0.11**	**3**
	Large	**0.665**	0.301	**0.344**	**2**	0.181	0.13	**0.112**	**0.062**	**1.4**	**0.021**	0.306	**3**
Family 5	Small	**0.643**	**0.234**	**0.349**	**2**	0.307	0.021	**0.112**	**0.062**	**1.4**	**0.021**	**0.104**	**3**
	Medium	**0.608**	**0.224**	**0.32**	**2**	0.301	0.035	**0.112**	**0.062**	**1.4**	**0.021**	**0.11**	**3**
	Large	**0.665**	0.301	**0.344**	**2**	0.181	0.13	**0.112**	**0.062**	**1.4**	**0.021**	0.306	**3**

#	= GVI & PFPF Suggest Commonality
#	= PFPF Suggests Additional Commonality
#	= Neither GVI or PFPF Suggest Commonality

Table 35: PFPF-Optimization Families that Most Closely Resemble GVI Recommendations

The five families with all of the commonality suggested by the GVI are shown as red points in Figure 91. The 243 families that differ in only one common parameter than recommended by the GVI are shown as green points. It is interesting to note the orientation of

all of these families within the tradespace. They are all clumped very near to the most common, most effective region of the tradespace.

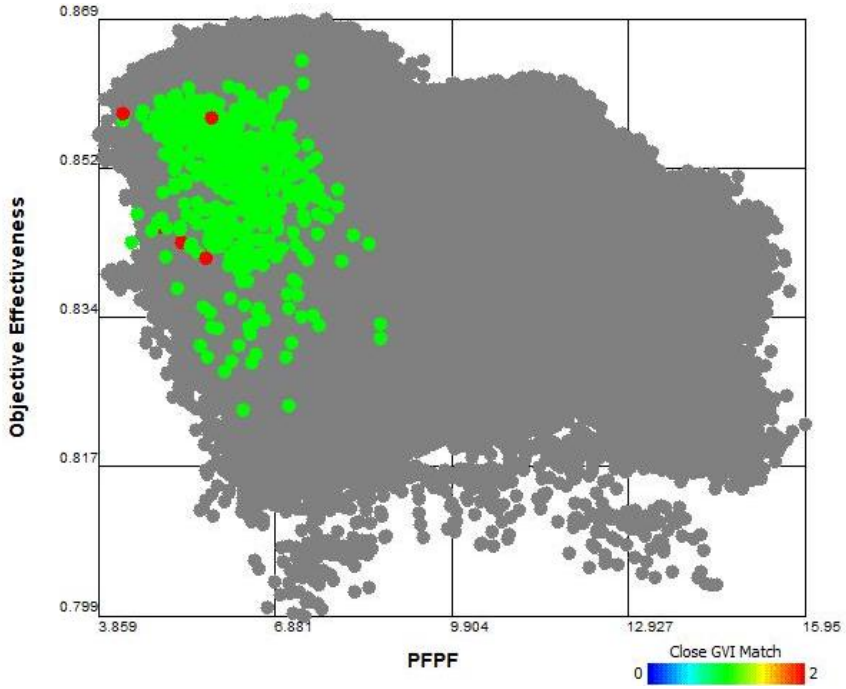

Figure 91: PFPF Families that Resemble GVI Analysis

Furthermore, plotting the objective effectiveness overall of all 729,000 families against PFPF and colored based on their similarity to the GVI recommendations for commonality the most similar families not only orient with lower PFPF values but also show a slight preference toward higher effectiveness as shown in Figure 92.

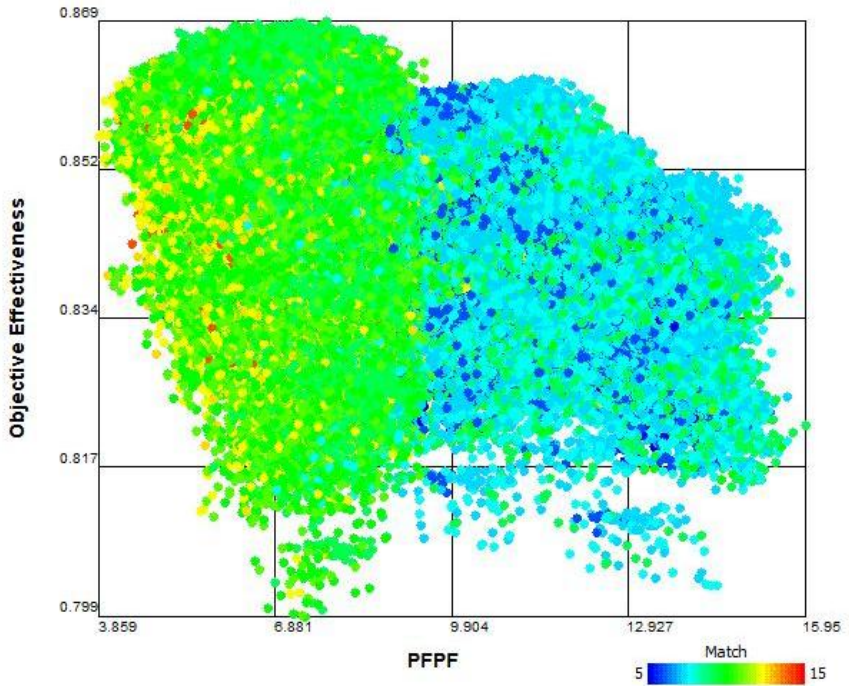

Figure 92: PFPF Families Colored Based on Similarity to GVI Recommendations

An important take-away from this analysis is that it is possible for experts completing GVI to suggest too much commonality, which may over-constrain the problem and limit the maximum effectiveness or performance of the family. While it may not match exactly, it is very likely for the results of the GVI to be similar to the family identified using PFPF as having the best tradeoff between commonality and performance. If the goal is to locate the family in the trade space that most closely matches GVI, then it may be beneficial to suggest commonality for a parameter when in doubt, and use optimization to examine the performance tradeoff from having that parameter common. In this regard, GVI and PFPF could be used effectively together to explore the trade space for the product family at hand.

Otherwise, more densely populating the trade space may result in the existence of several families matching the GVI suggestions for commonality. If this approach is taken, cost analysis and user needs can be considered during GVI analysis to ease the difficult decision of choosing

from several families with similar effectiveness and commonality. As seen in Figure 90 it can be very difficult to make a final selection between numerous families with very little differences in effectiveness and commonality. Thus, searching the trade space for families displaying similar commonality as suggested by the GVI can lead to a much easier decision. For the example being considered, the upper left red point in Figure 91 (see Family 2 in Table 35) benefits from all of the advantages to commonality discovered during GVI analysis yet still benefits from PFPF optimization. This family has a PFPF value of 4.291 and an objective effectiveness overall of 85.8%.

Finally, Table 35 may appear to indicate that PFPF proposes greater amounts of commonality than GVI; however, GVI is determined at the subsystem level whereas PFPF is calculated at the parametric level of each subsystem. For example, because GVI is binary at the modular level, it may say "batteries should be scalable" rather than determining which dimensions to make common while PFPF allows for scalability (e.g., "keep battery length and width common but scale height").

6.6 Summary of Optimization Method

This chapter presents the results of both the GVI analysis and PFPF optimization. Additionally, the results of both of these methods were compared. It was concluded from this comparison that there are a significant number of overlapping suggestions for commonality when comparing the results of GVI and PFPF. GVI may either over-constrain commonality or may miss opportunities for commonality. However, PFPF optimization does not benefit from expert opinion when making commonality decisions.

In order to maintain the benefits from both GVI and PFPF optimization a new method which combines both methods is suggested. This new method, which was first presented in Chapter 1, adheres to the following five steps:

(1) Perform GVI analysis

(2) Create a mathematical model of the product

(3) Perform individual PFPF optimization to populate the trade space

(4) Search the trade space for designs most closely resembling GVI commonality suggestions

(5) Choose the best design, that which has the highest effectiveness and lowest PFPF

It is possible to constrain commonality and create one family at a time. However, doing so has the potential to over-constrain the optimization problem. In some cases, the GVI may suggest commonality which presents a hindrance to the effectiveness of the individual robots. Rather, the method presented in this chapter will indicate to the designer when it is not possible to enforce all of the parameters suggested to be common by the GVI (i.e., the best family may lack one GVI suggested commonality parameter).

Chapter 7

Closing Remarks and Future Work

7.1 Summary of the Research

This work presents a mathematical model for determining geometry, sizing components, and calculating capabilities of a ground robot in Chapter 2 and 3. Chapter 4 presents validation of this model. Several tradeoffs in design parameters and capabilities were discovered and outlined in Chapter 5. Over fifteen thousand design iterations were generated using this model. These results were used to compare the results of the GVI to PFPF in Chapter 6.

The results of this product family trade study show that there are several opportunities for commonality within the robot family with minimal sacrifice in performance. Several common subsystems were identified using the GVI. Using the PFPF and optimization, three promising robot families were identified, including a Best Compromise Family that had a high Objective Overall Effectiveness (90%) and high degree of similarity among the parameter settings that defined the small, medium, and large robots in the family. It was shown that there can be a significant amount of similarity in the suggestions of GVI and PFPF optimization for commonality. If the ultimate goal is to maximize commonality where performance is not of a primary concern and a family member size is not specified as an output of the model (resulting in numerous family designs being invalid), then the method of starting with GVI for commonality may be a wise choice. Values for parameters that are known to be or desired to be common can then be evaluated. However, using all of the suggestions from GVI may add too many commonality constraints to the model and thus limit the maximum possible effectiveness of product family designs. The best method seems to be to first perform PFPF optimization and

then search the trade space for families that exhibit the closest commonality recommendations as the GVI.

7.2 Research Contributions

This work presents several unique contributions to the literature. First, the mathematical model presented in Chapters 2 and 3 is one of a kind in that it is a very fast, comprehensive, accurate (as shown in Chapter 4), yet simple model. Nearly all of the modeling efforts found in the existing literature involve very complex, multi-dimensional, computationally intensive algorithms. The model presented in Chapters 2 and 3 is necessary in order to be able to quickly and effectively perform trade studies such as those presented in Chapter 5. Such a model is also necessary in order to determine the best possible product family as discussed in Chapter 6. As previously mentioned, the comparison of PFPF to GVI is also unique to this work.

7.3 Future Work

The next step in this research is to perform more detailed analyses of these commonality recommendations to verify the results further. In addition, because some parameters may be harder to make common or some requirements may be more critical to mission success, which could be determined with GVI, the sensitivities of the weightings on (i) commonality and (ii) effectiveness must be understood in case prioritization or varying importance of weightings on parameters in future trade studies is requested. The results of this study are promising; however, more work is needed to better understand the role of expert opinion during product family optimization.

There are five robot families within the PFPF optimization trade space that have all of the same commonality suggested by the GVI. Therefore, the model will be reconfigured so that entire product families can be created with each design iteration.

Finally, the model needs to be validated further against a higher fidelity model. As previously mentioned, the higher fidelity model is not capable of sampling such a large portion of the trade space as quickly as this model; however, it is potentially capable of predicting more accurate results.

References

[1] Fellini, R., Kokkolaras, M. and Papalambros, P. Y., 2005, "Commonality Decisions in Product Family Design", *Product Platform and Product Family Design: Methods and Applications*, Simpson, T. W., Siddique, Z. and Jiao, J. Eds., New York, Springer, 157-185.

[2] Fixson, S. K., 2007, "Modularity and Commonality Research: Past Developments and Future Opportunities," *Concurrent Engineering: Research and Applications*, 15(2), 85-111.

[3] Thevenot, H. J. and Simpson, T. W., 2006, "Commonality Indices for Product Family Design: A Detailed Comparison," *Journal of Engineering Design*, 17(2), 99-119.

[4] Robertson, D. and Ulrich, K., 1998, "Planning Product Platforms," *Sloan Management Review*, 39(4), 19-31.

[5] Martin, M. V. and Ishii, K., 2002, "Design for Variety: Developing Standardized and Modularized Product Platform Architectures," *Research in Engineering Design*, 13(4), 213-235.

[6] Messac, A., Martinez, M. P. and Simpson, T. W., 2002, "A Penalty Function for Product Family Design Using Physical Programming," *ASME Journal of Mechanical Design*, 124(2), 164-172.

[7] Hauser, J. R. and Clausing, D., 1988, "The House of Quality," *Harvard Business Review*, 66(3), 63-73.

[8] Nomaguchi, Y., Taguchi, T. and Fujita, K., 2006, "Knowledge Model for Managing Product Variety and Its Reflective Design Process", *ASME Design Engineering Technical Conferences*, Philadelphia, PA, ASME, DETC2006-99360.

[9] Lee, H., Seol, H., Sung, N., Hong, Y. S. and Park, Y., 2010, "An Analytic Network Process Approach to Measuring Design Change Impacts in Modular Products," *Journal of Engineering Design*, 21(1), 75-91.

[10] Baldwin, C. Y. and Clark, K. B., 2000, *Design Rules: Volume 1. The Power of Modularity*, Cambridge, MA, MIT Press.

[11] Marshall, R., Leaney, P. G. and Botterell, P., 1998, "Enhanced Product Realization through Modular Design: An Example of Product/Process Integration," *Journal of Integrated Design and Process Technology*, 3(4), 143-150.

[12] Lee, H., Kim, C., Cho, H. and Park, Y., 2009, "An ANP-based Technology Network for Identification of Core Technologies: A Case of Telecommunication Technologies," *Expert Systems with Applications*, 36(1), 894-908.

[13] Yu, T.-L., Yassine, A. A. and Goldberg, D. E., 2007, "An Information Theoretic Method for Developing Modular Architectures Using Genetic Algorithms," *Research in Engineering Design*, 18(2), 91-109.

[14] Kang, C. M. and Hong, S.-K., 2009, "A Framework for Designing Balanced Product Platforms by Estimating the Versatility of Components," *International Journal of Production Research*, 47(19), 5271-5295.

[15] Takai, S. and Ishii, K., 2004, "Modifying Pugh's Design Concept Evaluation Methods", *ASME Design Engineering Technical Conferences - Design Theory & Methodology Conference*, Salt Lake City, UT, ASME, DETC2004/DTM-57512.

[16] Simpson, T. W., 2005, "Methods for Optimizing Product Platforms and Product Families: Overview and Classification", *Product Platform and Product Family Design: Methods*

and Applications, Simpson, T. W., Siddique, Z. and Jiao, J. Eds., New York, Springer, 133-156.

[17] Khajavirad, A. and Michalek, J., 2007, "An Extension of the Commonality Index for Product Family Optimization", *ASME Design Engineering Technical Conferences - Design Automation Conference*, Las Vegas, NV, ASME, DETC2007/DAC-35605.

[18] "Simulink - Simulation and Model-Based Design," Retrieved July 2 2010, 2010, from http://www.mathworks.com/products/simulink/.

[19] Stump, G. M., Yukish, M., Simpson, T. W. and Bennett, L., 2002, "Multidimensional Visualization and Its Application to a Design by Shopping Paradigm", *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, GA, AIAA, AIAA-2002-5622.

[20] Stump, G., Yukish, M. and Simpson, T. W., 2004, "The ARL Trade Space Visualizer: An Engineering Decision-Making Tool", *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Albany, NY, AIAA, AIAA-2004-4568.

[21] Madavan, N. K., 2002, "Multiobjective Optimization Using a Pareto Differential Evolution Approach", *Congress on Evolutionary Computation (CEC'2002)*, Piscataway, NJ, IEEE Service Center, 1145-1150.

[22] Simpson, T. W., Spencer, D. B., Yukish, M. A. and Stump, G., 2008, "Visual Steering Commands and Test Problems to Support Research in Trade Space Exploration", *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization* Conference, Victoria, British Columbia, Canada, AIAA, AIAA-2008-6085.

[23] Slingerland, L. A., Bobuk, A. and Simpson, T. W., 2010, "From User Requirements to Commonality Specifications: A Detailed Example of Product Family Design", *13th AIAA/ISSMO Multidisciplinary Analysis Optimization* Conference, Fort Worth, Texas, AIAA-2010.

[24] Inselberg, A. and Dimsdale, B., 1990, "Parallel Coordinates: A Tool For Visualizing Multi-Dimensional Geometry", *IEEE Visualization '90*, IEEE, 361-378.

[25] Khalil, H. K., 1996, *Nonlinear Systems*, Upper Saddle River, New Jersey, Prentice Hall.

[26] Logan, D., 2010, Mechanical Engineering. University Park, The Pennsylvania State University. Masters of Science.

[27] Andrew Pytel, J. K., 2003, *Mechanics Of Materials*, Pacific Grove, Brooks/Cole - Thomson Learning.

[28] Bosch, R. GmbH, Ed., 2007, *Automotive Electrics - Automotive Electronics*, Cambridge, MA, Bentley Publishers.

[29] Pop, V., Bergveld, H. J., Danilov, D., Regtien, P. P. L. and Notten, P. H. L., 2008, *Battery Management Systems: Accurate State-of-Charge Indication for Battery-Powered Applications*, Springer Science and Buisiness Media B.V.

[30] Clarkson, P. J., Simons, C. and Eckert, C., 2004, "Predicting Change Propagation in Complex Design," *ASME Journal of Mechanical Design*, 126(5), 788-797.

[31] Donaldson, B., Industrial Engineering. University Park, The Pennsylvania State University. M.S.

[32] Stump, G., Lego, S., Yukish, M., Simpson, T. W. and Donndelinger, J. A., 2009, "Visual Steering Commands for Trade Space Exploration: User-Guided Sampling with Example," *ASME Journal of Computing and Information Science in Engineering*, 9(4), 044501 (10 pgs).

Appendix A

Method to Create an Executable Simulink® Model

(1) Using a computer with Matlab® and the Real-Time Workshop toolbox installed open the Simulink® model.

(2) Open the script that imports values into the variables used in the model. Define each input value as something sequential yet random. For example, use **999**234231432, **998**324678234, **997**273650283, etc.

(3) Run the Matlab® script that imports the values specified in step 2 and pass them through the model. However, using these values may cause the model to crash, therefore, set a breakpoint to prevent the model from actually running.

(4) Create a Windows executable version of the Simulink® model. In the menu of the Simulink® model click on: Tools → Real Time Workshop → Build Model

(5) Type: param_struct = rsimgetrtp('robot_design_model_5') into the Matlab® workspace

(6) Search for the values specified in step 2 by typing: find(param_struct.parameters.values == 999234231432)

(7) Save the .mat file by typing: save RobotSimParams.mat param_struct

(8) Ensure that the Micorsoft Software Development Kit (SDK) is installed correctly by selecting a compiler:

   (a) First, in the command window type "mex –setup" and select your desired compiler.

   (b) Second, in the command window type "mbuild –setup" and select your desired compiler.

(c) If the SDK is installed incorrectly the "mbuild –setup" command results in an error. The SDK can be downloaded for free from Microsoft's website.

(9) Run a Matlab® script that does the following. See Appendix B for a copy of this script.

(a) Read in the model inputs to a vector named tinput from a text file

(b) Load the .mat file created in step 7.

(c) Pass the values from the text file from step 8a to the necessary locations in the .mat file by typing

(d) Save the .mat file

(e) Clear all variables

(f) Delete the old mat file created from the model

(g) Run the executable model

(h) Load the .mat file created by the model

(i) Write the output variables to a text file

(10) Launch the Matlab® deployment tool by typing "deploytool" into the command window.

(11) Name the file and specify a location. Be sure to select a standalone application.

(12) Add the main file which was created in step 9

(13) Add additional files: the executable model file created in step 4, the .mat files (RobotSimParams.mat and RobotSimParams.mat), input.txt and output.txt

(14) If the executable model needs to be run on another computer, be sure to include the library files.

(15) Click on build package.

(16) Double clicking on the .exe file installs the library files if selected, as well as unpack all of the additional files and run the simulation.

Appendix B

Matlab® Script to Deploy Executable Package

```
%-------------------------------------------------------------------------
% Created by Aaron Bobuk
% Last Updated 7/14/2010

% This script is used to package the simulation using the
% deployment tool in Matlab to create an executable model

%-------------------------------------------------------------------------


% Read in a text file from ATSV
% ATSV required a format of: variable1, value
%                            variable2, value   etc
fid = fopen('inputs.txt','r');
S = textscan(fid,'%s %f','delimiter',',');

% Define tinput vector as the second column
tinput = S{2};

% Close the text file
fclose(fid);


%-------------------------------------------------------------------------

% Place the values of the input variables in the correct
% locations within the .mat file

load RobotSimParams.mat;
param_struct.parameters.values([101, 153]) = tinput(1);
param_struct.parameters.values([102, 157]) = tinput(2);
param_struct.parameters.values([108, 154]) = tinput(3);
param_struct.parameters.values([114, 155]) = tinput(4);
param_struct.parameters.values([120, 156]) = tinput(5);
param_struct.parameters.values([107]) = tinput(6);
param_struct.parameters.values([105]) = tinput(7);
param_struct.parameters.values([113]) = tinput(8);
param_struct.parameters.values([119]) = tinput(9);
param_struct.parameters.values([109]) = tinput(10);
param_struct.parameters.values([115]) = tinput(11);
param_struct.parameters.values([121]) = tinput(12);
param_struct.parameters.values([83, 103, 150, 151]) = tinput(13);
param_struct.parameters.values([100]) = tinput(14);
param_struct.parameters.values([110]) = tinput(15);
param_struct.parameters.values([116]) = tinput(16);
param_struct.parameters.values([122]) = tinput(17);
param_struct.parameters.values([80, 117, 152]) = tinput(18);
param_struct.parameters.values([177]) = tinput(19);
```

```
param_struct.parameters.values([178]) = tinput(20);
param_struct.parameters.values([1]) = tinput(21);
param_struct.parameters.values([8, 56, 128]) = tinput(22);
param_struct.parameters.values([15]) = tinput(23);
param_struct.parameters.values([22, 26, 27, 124]) = tinput(24);
param_struct.parameters.values([29, 32]) = tinput(25);
param_struct.parameters.values([28, 31, 34]) = tinput(26);
param_struct.parameters.values([55,59,61,69,78,84,145,158]) = tinput(27);
param_struct.parameters.values([36]) = tinput(28);
param_struct.parameters.values([37]) = tinput(29);
param_struct.parameters.values([25, 98]) = tinput(30);
param_struct.parameters.values([60]) = tinput(31);
param_struct.parameters.values([82, 99, 142, 149]) = tinput(32);
param_struct.parameters.values([63]) = tinput(33);
param_struct.parameters.values([62, 73]) = tinput(34);
param_struct.parameters.values([65, 70]) = tinput(35);
param_struct.parameters.values([64]) = tinput(36);
param_struct.parameters.values([23, 68, 147, 148]) = tinput(37);
param_struct.parameters.values([33, 81]) = tinput(38);
param_struct.parameters.values([143]) = tinput(39);
param_struct.parameters.values([144]) = tinput(40);
param_struct.parameters.values([146]) = tinput(41);
param_struct.parameters.values([159]) = tinput(42);


% Save the .mat file created above
save RobotSimParams.mat param_struct;

%--------------------------------------------------------------------------

% Clear all variables
clear all;

% Delete the old .mat file created by the model
delete('robot_design_model_5.mat');

% Run the Simulation
system('robot_design_model_5 -p RobotSimParams.mat');

%--------------------------------------------------------------------------



% Load the .mat file created by the model
load('robot_design_model_5.mat');

% Determine the last value in the vector output by the model
lastIteration = length(rt_MotorL1A1PeakPower);

% Initialize the output vector
toutput = zeros(123,1);

% Delete the old output file
% NOTE: you must have a file called outputs when you start
delete('soutputs.txt')
```

```
% Create a new output txt file
fid = fopen('soutputs.txt','a');


%-------------------------------------------------------------------------


% Write the final values to a text file
%-----------------------------Manipulator----------------------------------
toutput(1) = rt_MotorL1A1PeakPower(lastIteration);
    fprintf(fid,'%s, %f \n','MotorL1A1PeakPower',toutput(1));
toutput(2) = rt_MotorL2A1PeakPower(lastIteration);
    fprintf(fid,'%s, %f \n','MotorL2A1PeakPower',toutput(2));
toutput(3) = rt_MotorL3A1PeakPower(lastIteration);
    fprintf(fid,'%s, %f \n','MotorL3A1PeakPower',toutput(3));
toutput(4) = rt_MotorL4A1PeakPower(lastIteration);
    fprintf(fid,'%s, %f \n','MotorL4A1PeakPower',toutput(4));
toutput(5) = rt_L1A1SpeedMaxEff(lastIteration);
    fprintf(fid,'%s, %f \n','L1A1SpeedMaxEff',toutput(5));
toutput(6) = rt_L2A1SpeedMaxEff(lastIteration);
    fprintf(fid,'%s, %f \n','L2A1SpeedMaxEff',toutput(6));
toutput(7) = rt_L3A1SpeedMaxEff(lastIteration);
    fprintf(fid,'%s, %f \n','L3A1SpeedMaxEff',toutput(7));
toutput(8) = rt_L4A1SpeedMaxEff(lastIteration);
    fprintf(fid,'%s, %f \n','L4A1SpeedMaxEff',toutput(8));
toutput(9) = rt_L1A1TotalMotorMass(lastIteration);
    fprintf(fid,'%s, %f \n','L1A1TotalMotorMass',toutput(9));
toutput(10) = rt_L2A1TotalMotorMass(lastIteration);
    fprintf(fid,'%s, %f \n','L2A1TotalMotorMass',toutput(10));
toutput(11) = rt_L3A1TotalMotorMass(lastIteration);
    fprintf(fid,'%s, %f \n','L3A1TotalMotorMass',toutput(11));
toutput(12) = rt_L4A1TotalMotorMass(lastIteration);
    fprintf(fid,'%s, %f \n','L4A1TotalMotorMass',toutput(12));
toutput(13) = rt_L1A1torque(lastIteration);
    fprintf(fid,'%s, %f \n','L1A1torque',toutput(13));
toutput(14) = rt_L2A1torque(lastIteration);
    fprintf(fid,'%s, %f \n','L2A1torque',toutput(14));
toutput(15) = rt_L3A1torque(lastIteration);
    fprintf(fid,'%s, %f \n','L3A1torque',toutput(15));
toutput(16) = rt_L4A1torque(lastIteration);
    fprintf(fid,'%s, %f \n','L4A1torque',toutput(16));
toutput(17) = rt_L1A1radius(lastIteration);
    fprintf(fid,'%s, %f \n','L1A1radius',toutput(17));
toutput(18) = rt_L2A1radius(lastIteration);
    fprintf(fid,'%s, %f \n','L2A1radius',toutput(18));
toutput(19) = rt_L3A1radius(lastIteration);
    fprintf(fid,'%s, %f \n','L3A1radius',toutput(19));
toutput(20) = rt_totalDOF(lastIteration);
    fprintf(fid,'%s, %f \n','totalDOF',toutput(20));
toutput(21) = rt_TotalManipulatorMass(lastIteration);
    fprintf(fid,'%s, %f \n','TotalManipulatorMass',toutput(21));
toutput(22) = rt_MaxManipulatorPowerDraw(lastIteration);
    fprintf(fid,'%s, %f \n','MaxManipulatorPowerDraw',toutput(22));
%-----------------------------Batteries------------------------------------
toutput(23) = rt_peukertNumber(lastIteration);
```

```
    fprintf(fid,'%s, %f \n','peukertNumber',toutput(23));
toutput(24) = rt_batteryLength(lastIteration);
    fprintf(fid,'%s, %f \n','batteryLength',toutput(24));
toutput(25) = rt_batteryWidth(lastIteration);
    fprintf(fid,'%s, %f \n','batteryWidth',toutput(25));
toutput(26) = rt_batteryHeight(lastIteration);
    fprintf(fid,'%s, %f \n','batteryHeight',toutput(26));
toutput(27) = rt_batteryVolume(lastIteration);
    fprintf(fid,'%s, %f \n','batteryVolume',toutput(27));
toutput(28) = rt_totalBatteryMass(lastIteration);
    fprintf(fid,'%s, %f \n','totalBatteryMass',toutput(28));
toutput(29) = rt_totalBatteryPower(lastIteration);
    fprintf(fid,'%s, %f \n','totalBatteryPower',toutput(29));
toutput(30) = rt_numberOfBatteries(lastIteration);
    fprintf(fid,'%s, %f \n','numberOfBatteries',toutput(30));
toutput(31) = rt_batteryCellCapacityWithDOD(lastIteration);
    fprintf(fid,'%s, %f \n','batteryCellCapacityWithDOD',toutput(31));
toutput(32) = rt_manipulatorPowerFlag(lastIteration);
    fprintf(fid,'%s, %f \n','manipulatorPowerFlag',toutput(32));
%-----------------------------Motor Controller-----------------------------
toutput(33) = rt_auxPowerFlag(lastIteration);
    fprintf(fid,'%s, %f \n','auxPowerFlag',toutput(33));
toutput(34) = rt_motorControllerMass(lastIteration);
    fprintf(fid,'%s, %f \n','motorControllerMass',toutput(34));
toutput(35) = rt_motorControllerLength(lastIteration);
    fprintf(fid,'%s, %f \n','motorControllerLength',toutput(35));
toutput(36) = rt_motorControllerWidth(lastIteration);
    fprintf(fid,'%s, %f \n','motorControllerWidth',toutput(36));
toutput(37) = rt_motorControllerHeight(lastIteration);
    fprintf(fid,'%s, %f \n','motorControllerHeight',toutput(37));
toutput(38) = rt_motorControllerMaxEfficiency(lastIteration);
    fprintf(fid,'%s, %f \n','motorControllerMaxEfficiency',toutput(38));
toutput(39) = rt_motorControllerEfficiency(lastIteration);
    fprintf(fid,'%s, %f \n','motorControllerEfficiency',toutput(39));
%-----------------------------Drive Motor----------------------------------
toutput(40) = rt_motorConstantK(lastIteration);
    fprintf(fid,'%s, %f \n','motorConstantK',toutput(40));
toutput(41) = rt_motorMass(lastIteration);
    fprintf(fid,'%s, %f \n','motorMass',toutput(41));
toutput(42) = rt_motorLength(lastIteration);
    fprintf(fid,'%s, %f \n','motorLength',toutput(42));
toutput(43) = rt_motorDiameter(lastIteration);
    fprintf(fid,'%s, %f \n','motorDiameter',toutput(43));
toutput(44) = rt_noLoadCurrent(lastIteration);
    fprintf(fid,'%s, %f \n','noLoadCurrent',toutput(44));
toutput(45) = rt_armResistance(lastIteration);
    fprintf(fid,'%s, %f \n','armResistance',toutput(45));
toutput(46) = rt_motorRefVoltage(lastIteration);
    fprintf(fid,'%s, %f \n','motorRefVoltage',toutput(46));
toutput(47) = rt_motorStallTorque(lastIteration);
    fprintf(fid,'%s, %f \n','motorStallTorque',toutput(47));
toutput(48) = rt_motorContTorque(lastIteration);
    fprintf(fid,'%s, %f \n','motorContTorque',toutput(48));
toutput(49) = rt_motorMaxEff(lastIteration);
    fprintf(fid,'%s, %f \n','motorMaxEff',toutput(49));
toutput(50) = rt_motorNoLoadOmega(lastIteration);
    fprintf(fid,'%s, %f \n','motorNoLoadOmega',toutput(50));
```

```
toutput(51) = rt_gearRatioEff(lastIteration);
    fprintf(fid,'%s, %f \n','gearRatioEff',toutput(51));
toutput(52) = rt_outputEff(lastIteration);
    fprintf(fid,'%s, %f \n','outputEff',toutput(52));
toutput(53) = rt_maxOutputTorque(lastIteration);
    fprintf(fid,'%s, %f \n','maxOutputTorque',toutput(53));
toutput(54) = rt_maxSteadyStateTorque(lastIteration);
    fprintf(fid,'%s, %f \n','maxSteadyStateTorque',toutput(54));
toutput(55) = rt_gearRatio(lastIteration);
    fprintf(fid,'%s, %f \n','gearRatio',toutput(55));
toutput(56) = rt_numberMotors(lastIteration);
    fprintf(fid,'%s, %f \n','numberMotors',toutput(56));
toutput(57) = rt_gearboxMass(lastIteration);
    fprintf(fid,'%s, %f \n','gearboxMass',toutput(57));
toutput(58) = rt_totalDriveMotorMass(lastIteration);
    fprintf(fid,'%s, %f \n','totalDriveMotorMass',toutput(58));
%-----------------------------Chassis Dimensions--------------------------
toutput(59) = rt_chassisWidth(lastIteration);
    fprintf(fid,'%s, %f \n','chassisWidth',toutput(59));
toutput(60) = rt_vehicleLength(lastIteration);
    fprintf(fid,'%s, %f \n','vehicleLength',toutput(60));
toutput(61) = rt_chassisHeight(lastIteration);
    fprintf(fid,'%s, %f \n','chassisHeight',toutput(61));
%-----------------------------Structure-----------------------------------
toutput(62) = rt_finalChassisHeight(lastIteration);
    fprintf(fid,'%s, %f \n','finalChassisHeight',toutput(62));
toutput(63) = rt_chassisMass(lastIteration);
    fprintf(fid,'%s, %f \n','chassisMass',toutput(63));
toutput(64) = rt_finalDeflection(lastIteration);
    fprintf(fid,'%s, %f \n','finalDeflection',toutput(64));
%-----------------------------Wheels/Tracks-------------------------------
toutput(65) = rt_wheelbase(lastIteration);
    fprintf(fid,'%s, %f \n','wheelbase',toutput(65));
toutput(66) = rt_massWheelTrack(lastIteration);
    fprintf(fid,'%s, %f \n','massWheelTrack',toutput(66));
toutput(67) = rt_massWheelsForTrack(lastIteration);
    fprintf(fid,'%s, %f \n','massWheelsForTrack',toutput(67));
toutput(68) = rt_widthWheelTrack(lastIteration);
    fprintf(fid,'%s, %f \n','widthWheelTrack',toutput(68));
toutput(69) = rt_widthSteerWheels(lastIteration);
    fprintf(fid,'%s, %f \n','widthSteerWheels',toutput(69));
toutput(70) = rt_heightTrack(lastIteration);
    fprintf(fid,'%s, %f \n','heightTrack',toutput(70));
toutput(71) = rt_contactLengthFlagged(lastIteration);
    fprintf(fid,'%s, %f \n','contactLengthFlagged',toutput(71));
toutput(72) = rt_wheelDiameterFlag(lastIteration);
    fprintf(fid,'%s, %f \n','wheelDiameterFlag',toutput(72));
%-----------------------------PowerReq------------------------------------
toutput(73) = rt_maxCruiseVelocity(lastIteration);
    fprintf(fid,'%s, %f \n','maxCruiseVelocity',toutput(73));
%-----------------------------Endurance-----------------------------------
toutput(74) = rt_batteryTime(lastIteration);
    fprintf(fid,'%s, %f \n','batteryTime',toutput(74));
toutput(75) = rt_batteryDistance(lastIteration);
    fprintf(fid,'%s, %f \n','batteryDistance',toutput(75));
toutput(76) = rt_batteryVelocityFlag(lastIteration);
    fprintf(fid,'%s, %f \n','batteryVelocityFlag',toutput(76));
```

```
%-----------------------------Total Vehicle Dimensions---------------------
toutput(77) = rt_vehicleWidth(lastIteration);
    fprintf(fid,'%s, %f \n','vehicleWidth',toutput(77));
toutput(78) = rt_CGy(lastIteration);
    fprintf(fid,'%s, %f \n','CGy',toutput(78));
toutput(79) = rt_CGx(lastIteration);
    fprintf(fid,'%s, %f \n','CGx',toutput(79));
toutput(80) = rt_CGz(lastIteration);
    fprintf(fid,'%s, %f \n','CGz',toutput(80));
toutput(81) = rt_vehicleHeight(lastIteration);
    fprintf(fid,'%s, %f \n','vehicleHeight',toutput(81));
toutput(82) = rt_totalMotorTorqueMax(lastIteration);
    fprintf(fid,'%s, %f \n','totalMotorTorqueMax',toutput(82));
toutput(83) = rt_CGxSAE(lastIteration);
    fprintf(fid,'%s, %f \n','CGxSAE',toutput(83));
toutput(84) = rt_CGySAE(lastIteration);
    fprintf(fid,'%s, %f \n','CGySAE',toutput(84));
toutput(85) = rt_CGzSAE(lastIteration);
    fprintf(fid,'%s, %f \n','CGzSAE',toutput(85));
%-----------------------------Vehice Mass----------------------------------
toutput(86) = rt_vehicleMass(lastIteration);
    fprintf(fid,'%s, %f \n','vehicleMass',toutput(86));
%-----------------------------Functional Capabilities----------------------
toutput(87) = rt_thetaMaxSlopeClimb(lastIteration);
    fprintf(fid,'%s, %f \n','thetaMaxSlopeClimb',toutput(87));
toutput(88) = rt_thetaMaxSlopeClimbLimitingFactor(lastIteration);
    fprintf(fid,'%s, %f \n','thetaMaxSlopeClimbLimitingFactor',toutput(88));
toutput(89) = rt_thetaMaxTraverse(lastIteration);
    fprintf(fid,'%s, %f \n','thetaMaxTraverse',toutput(89));
toutput(90) = rt_thetaMaxTraverseLimitingFactor(lastIteration);
    fprintf(fid,'%s, %f \n','thetaMaxTraverseLimitingFactor',toutput(90));
toutput(91) = rt_heightStepface(lastIteration);
    fprintf(fid,'%s, %f \n','heightStepface',toutput(91));
toutput(92) = rt_thetaMaxCurbLimitingFactor(lastIteration);
    fprintf(fid,'%s, %f \n','thetaMaxCurbLimitingFactor',toutput(92));
toutput(93) = rt_thetaMaxCurb(lastIteration);
    fprintf(fid,'%s, %f \n','thetaMaxCurb',toutput(93));
toutput(94) = rt_heightStep(lastIteration);
    fprintf(fid,'%s, %f \n','heightStep',toutput(94));
toutput(95) = rt_LengthStair(lastIteration);
    fprintf(fid,'%s, %f \n','LengthStair',toutput(95));
toutput(96) = rt_canSpanMultiSteps(lastIteration);
    fprintf(fid,'%s, %f \n','canSpanMultiSteps',toutput(96));
toutput(97) = rt_HStairSpan(lastIteration);
    fprintf(fid,'%s, %f \n','HStairSpan',toutput(97));
toutput(98) = rt_thetaStairSpan(lastIteration);
    fprintf(fid,'%s, %f \n','thetaStairSpan',toutput(98));
toutput(99) = rt_LStairSpan(lastIteration);
    fprintf(fid,'%s, %f \n','LStairSpan',toutput(99));
toutput(100) = rt_maxDitchWidth(lastIteration);
    fprintf(fid,'%s, %f \n','maxDitchWidth',toutput(100));
toutput(101) = rt_torqueMotorAsphalt(lastIteration);
    fprintf(fid,'%s, %f \n','torqueMotorAsphalt',toutput(101));
toutput(102) = rt_zeroTurnRadiusAsphalt(lastIteration);
    fprintf(fid,'%s, %f \n','zeroTurnRadiusAsphalt',toutput(102));
toutput(103) = rt_torqueMotorGrass(lastIteration);
    fprintf(fid,'%s, %f \n','torqueMotorGrass',toutput(103));
```

```
toutput(104) = rt_zeroTurnRadiusGrass(lastIteration);
    fprintf(fid,'%s, %f \n','zeroTurnRadiusGrass',toutput(104));
toutput(105) = rt_WHallway(lastIteration);
    fprintf(fid,'%s, %f \n','WHallway',toutput(105));
toutput(106) = rt_HallwayWidthTurn(lastIteration);
    fprintf(fid,'%s, %f \n','HallwayWidthTurn',toutput(106));
toutput(107) = rt_HallwayWidthDoorway(lastIteration);
    fprintf(fid,'%s, %f \n','HallwayWidthDoorway',toutput(107));
toutput(108) = rt_maxDraggingMassTile(lastIteration);
    fprintf(fid,'%s, %f \n','maxDraggingMassTile',toutput(108));
toutput(109) = rt_maxDraggingMassConcrete(lastIteration);
    fprintf(fid,'%s, %f \n','maxDraggingMassConcrete',toutput(109));
toutput(110) = rt_maxDraggingMassGrass(lastIteration);
    fprintf(fid,'%s, %f \n','maxDraggingMassGrass',toutput(110));
toutput(111) = rt_maxDraggingMassAsphalt(lastIteration);
    fprintf(fid,'%s, %f \n','maxDraggingMassAsphalt',toutput(111));
toutput(112) = rt_maxDraggingMassGravel(lastIteration);
    fprintf(fid,'%s, %f \n','maxDraggingMassGravel',toutput(112));
toutput(113) = rt_maxDraggingMassDirt(lastIteration);
    fprintf(fid,'%s, %f \n','maxDraggingMassDirt',toutput(113));
toutput(114) = rt_maxDraggingMassClay(lastIteration);
    fprintf(fid,'%s, %f \n','maxDraggingMassClay',toutput(114));
toutput(115) = rt_rubbleDiameter(lastIteration);
    fprintf(fid,'%s, %f \n','rubbleDiameter',toutput(115));
toutput(116) = rt_thetaMaxStepfaceFrictionFlag(lastIteration);
    fprintf(fid,'%s, %f \n','thetaMaxStepfaceFrictionFlag',toutput(116));
%-----------------------------Manipulator Capabilities----------------------
toutput(117) = rt_selfRightingCapability(lastIteration);
    fprintf(fid,'%s, %f \n','selfRightingCapability',toutput(117));
toutput(118) = rt_manipulatorLength(lastIteration);
    fprintf(fid,'%s, %f \n','manipulatorLength',toutput(118));
toutput(119) = rt_farReachDistanceNoLift(lastIteration);
    fprintf(fid,'%s, %f \n','farReachDistanceNoLift',toutput(119));
toutput(120) = rt_farReachDistanceWithLift(lastIteration);
    fprintf(fid,'%s, %f \n','farReachDistanceWithLift',toutput(120));
toutput(121) = rt_cantLiftFLiftFlag(lastIteration);
    fprintf(fid,'%s, %f \n','cantLiftFLiftFlag',toutput(121));
%-----------------------------Effectiveness---------------------------------
toutput(122) = rt_size(lastIteration,1);
    fprintf(fid,'%s, %f \n','size',toutput(122));
toutput(123) = rt_effectiveness(lastIteration,1);
    fprintf(fid,'%s, %f \n','effectiveness',toutput(123));
toutput(124) = rt_thresholdEffectiveness(lastIteration,1);
    fprintf(fid,'%s, %f \n','thresholdEffectiveness',toutput(124));
toutput(125) = rt_objectiveEffectiveness(lastIteration,1);
    fprintf(fid,'%s, %f \n','objectiveEffectiveness',toutput(125));


% Close the .txt file
status=fclose(fid);
```