Virtual Vehicle Analysis and Simulation Environment

User's Guide

Kerry Loux

version 0.12a

v0.8b: 29 July 2009
v0.12a: 30 July 2015

Contents

Preface

I first became interested in vehicle dynamics through involvement with my college Formula SAE team. I was on summer break after finishing my freshman year, and somehow I stumbled upon the team's website. Racecars! Cool! So the next weekend, after some e-mails back and forth, I set out to a local autocross where the team was cooking burgers and hot dogs as a fundraiser. Cars of all kinds racing through a sea of orange cones. Having never been to an autocross before, I was curious. "Why is it that some cars lift the front wheel in the corners, and others lift the rear wheels?" "How come the real racecars don't screech tires like the street cars do?"

It didn't take long before it was clear that I had caught the racecar bug. I was particularly drawn to suspensions. I'm not entirely sure why that was, maybe because you could see all of the moving parts, or maybe because there was an experienced alumni who spent a lot of time in the shop who was also interested in suspension design. Anyway, we spent many hours discussing vehicle dynamics and suspension design, pouring through the team's library of reference material and studying suspension kinematics with the help of purpose-built commercially available software.

The software we were using had terrible graphics and was somewhat tedious to use, but it provided accurate results and was simple enough to be confident that we provided the correct input. It only permitted kinematic analysis of the suspension, and only one end of the car at a time, but it seemed well suited to our needs. As I gained more experience and started to search for justification of more design decisions, it became clear that this software had some shortcomings. At the time, it was our only option and many of the other engineering packages we used also had shortcomings. Most of the time, it was easy to accept the issues and work around them.

In the summer of 2005, immediately following the conclusion of the 2005 FSAE season, a good friend of mine, Matt Jarvis, began work on redesigning the frame to be stiffer and lighter. Matt was (and still is) a brilliant engineer. Among his interests were programming, finite element analysis and composites. He had taken the lead on the bodywork the previous year and wanted to consider adding composite panels to the frame design. We had previously used aluminum panels for this purpose, bonding and riveting them to the frame. Matt knew that a composite solution could improve the design, but it introduced several variables that we hadn't had to consider before.

Traditionally, someone would create a finite element model of the frame and apply a virtual torsional load. The displacement was calculated by the finite element software from which a stiffness could be calculated. The stiffness was compared to the weight to determine the overall quality of the design. Then the designer would look at the stresses in the frame members and adjust the geometry and/or the cross section of certain members, with the goal of improving stiffness and reducing weight. This process would continue until the designer stopped making progress or ran out of time. Then they'd just pick the frame that had the best stiffness to weight ratio, and start fabrication.

Matt looked at this problem and saw all of the available permutations. Even if you assumed the geometry was fixed, the number of frame members multiplied with the number of available cross sections (round, square, rectangular, dimensions, wall thickness) meant there was an incredible number of possible combinations. He wrote VBA macros into excel spreadsheets that implemented a genetic search algorithm to intelligently evaluate a huge number of these permutations and hopefully converge on a solution. After commandeering the computer lab overnight to run the optimization in parallel on some 20 machines, nearly 100,000 frame designs had been simulated. The best design was both stiffer and lighter than the best human-designed frame from the previous year. I was incredibly impressed by his success.

Also for the 2005-06 FSAE season, we decided to set the springs and dampers side-by-side instead of using coil-overs in an attempt to reduce the bending load on the damper shafts. In theory, this would reduce friction in the damper and give us more control over the damping characteristics of the suspension. This meant spending twice as much effort at each end to ensure the bellcrank geometry gave us the desired installation ratios across the entire range of motion. Unfortunately, this was also one of the more tedious jobs to tackle with our suspension kinematics software. Ideally, the designer would be able to look at the bellcrank along it's pivot axis, giving a clear view of the angles between the pushrod, bellcrank and spring or damper, and how they change through the range of motion. Or better yet - show a plot of the installation ratio across the desired range of motion, updated dynamically as the bellcrank design was adjusted. Also, it is important that all of these points lie on a common plane. Our suspension package did not permit either of the above. Our work-around was to bounce back-and-forth between our CAD software and the suspension design software. This was not going to be a fun task.

After spending several hours trying to work out the bellcrank geometry, I thought I would try to solve the kinematics problem myself, creating a tool that was designed exactly for the problem at hand. A few hours later, I had written some VBA code in an Excel spreadsheet that elegantly solved the problem of designing bellcrank geometry. Even with the time invested in the creation of the tool, the total time to design new bellcranks had been reduced and the results were probably better than what would have been achieved with our prior workflow. I experienced a great feeling of satisfaction, attached more to

Figure 1: The 2004 and (incomplete) 2005 Drexel University entries into the Formula SAE competitions

the creation of a useful tool than completing the design of the bellcranks.

Following my success with my bellcrank spreadsheet, I became interested in attempting to simulate the dynamics of the entire racecar. With good tire data and mass properties data, we could simulate the car performing maneuvers and quantify how changes in *anything* would affect lap times. In December 2005 I took a stab at writing software that would model a full vehicle - tires, suspension, engine, brakes, etc. I was extremely naive and this effort stopped only weeks after it started. It would have been a much better use of my time to focus on the manufacturing of our racecar (apologies to my 2005-06 team members!).

After graduating in 2007, I began work on what became the first incarnation of VVASE. It was a simple application that could solve the vehicle kinematics problem for double A-arm suspensions. I had made some improvements over the bellcrank spreadsheet I created a couple years earlier - it was faster, more accurate and more robust. But it didn't do anything more than what other commercially available suspension kinematics software could do - in fact, in most ways it did less. It was a learning exercise for me, but it wasn't very valuable as a tool.

At some point, I thought back to the genetic search algorithm that Matt had used to optimize the frame design. Could something similar be applied to suspension kinematics? After some careful thought and many hours of programming, I finally had something that was unique. As far as I knew, there was no commercially available suspension kinematics package that could do an optimization like VVASE.

Although I never pursued a job in the auto industry, my FSAE experience became a significant influence on my career. In addition to my engineering tasks associated with delivering our products, I develop special-purpose software tools for solving problems unique to our products and workflows. I still have an interest in vehicle dynamics and racing, but I was surprised to find that I have a passion for creating tools. This passion is what has driven me to continue work on VVASE, years after it began as an exercise and an experiment. I hope that it satisfies other's needs to analyze suspension kinematics

in the same manner it has satisfied my desire to create a tool.

Kerry R. Loux
Langhorne, PA
July 2015

Introduction

VVASE is a tool for analyzing suspension kinematics. Given the locations of the suspension linkages and the kinematic state of the vehicle (pitch, roll, heave and steer), many output values are computed. These outputs are key indicators of suspension performance and vehicle handling qualities.

There are other tools available that perform similar functions, but VVASE was designed with two unique features that make it stand out.

First, VVASE was designed to ease the process of evaluating differences between design options. Always visible is a panel that displays the differences between all open car files for the specified kinematic state. Additionally, iteration files can be used to compare kinematic outputs across a range of kinematic states - simultaneously for as many car files as desired. The iterations and panel displaying the kinematic outputs update continuously as suspension hardpoints are varied.

Second, VVASE provides a unique facility for performing complex optimizations. These optimizations are based on a genetic search algorithm that allows the user complete control over the optimization process. Genetic search algorithms allow broad coverage of highly non-linear solution spaces and can help avoid getting trapped at local minima. In suspension design, moving a single hardpoint can affect many output parameters. An optimization tool that balances multiple constraints is invaluable.

Currently, VVASE is limited to cars employing a double A-arm suspension at both ends. It supports several spring/damper attachment methods, U-bar and T-bar anti-roll bars, asymmetric suspensions and several other configuration options that allow virtually any double A-arm suspension to be modeled.

Chapter 1

Getting Started

1.1 Installation

VVASE does not include any installation process or changes to the machine registry. Simply place the executable and the pdf manual into the same directory. After running the first time, any changes to the default settings are saved to a configuration file which will appear in the directory from which the application is run.

1.2 Overview

Upon starting VVASE for the first time, the screen will look like Fig. 1.1. Counter-clockwise from the top left corner of the window are the *Systems Tree* (§ 1.2.1), *Edit Panel* (§ 1.2.2), *Output Pane* (§ 1.2.3), *Output List* (§ 1.2.4) and *Main Display Area* (§ 1.2.5). Above these elements are two toolbars, the *Kinematics Toolbar* (§ 1.2.6) and the *3D Toolbar* (§ 1.2.7). Across the top of the window is a typical menu bar, the contents of which are described in § 1.2.8. All of these elements can be dragged, floated or docked to create the desired layout.

1.2.1 Systems Tree

The *Systems Tree* lists all of the files that are currently open. Some files include sub-items which can be accessed by expanding the parent item. Right-clicking on *Systems Tree* entries results in a context menu from which files can be saved or closed.

1.2.2 Edit Panel

The *Edit Panel* is where the core modifications to all files are made. When a file is selected by either selecting its tab in the *Main Display Area* or clicking on it in the *Systems Tree*, the *Edit Panel* changes to show options relevant to the selected file. For files that include sub-items in the *Systems Tree*, each sub-item may have different editable parameters.

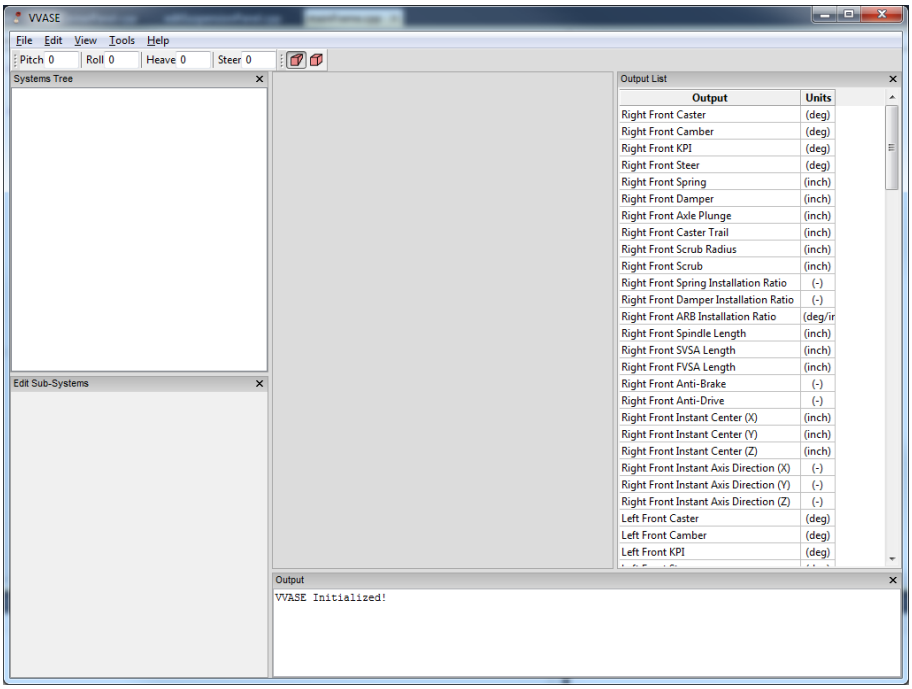

Figure 1.1: Default screen configuration

1.2.3 Output Pane

The *Output Pane* primarily contains diagnostic output to inform the user about problems encountered while computing the kinematics. For example, if a kinematic state is specified that would result in the suspension reaching its droop limit, the solver will be unable to solve for some of the hardpoint locations. Depending on how verbose the messages are configured to be (see § 1.3.4), the user will be notified that there was a problem computing the solution for one corner of the car or exactly which hardpoint it was that presented a problem.

1.2.4 Output List

The *Output List* contains a complete listing of all of the computed kinematic outputs for all open car files for the specified kinematic state (see § 1.2.6). The computed outputs are listed in rows and the open car files are listed in columns. This arrangement makes side-by-side comparisons of kinematic outputs for multiple car files very easy. When an output is undefined (e.g. when the kinematic roll center crosses the ground plane), the corresponding grid cell is highlighted yellow and the text "Undef." is displayed in the cell. If an output doesn't apply for a certain car file (e.g. front-wheel axle plunge on a rear-wheel drive car), the text "N/A" is displayed.

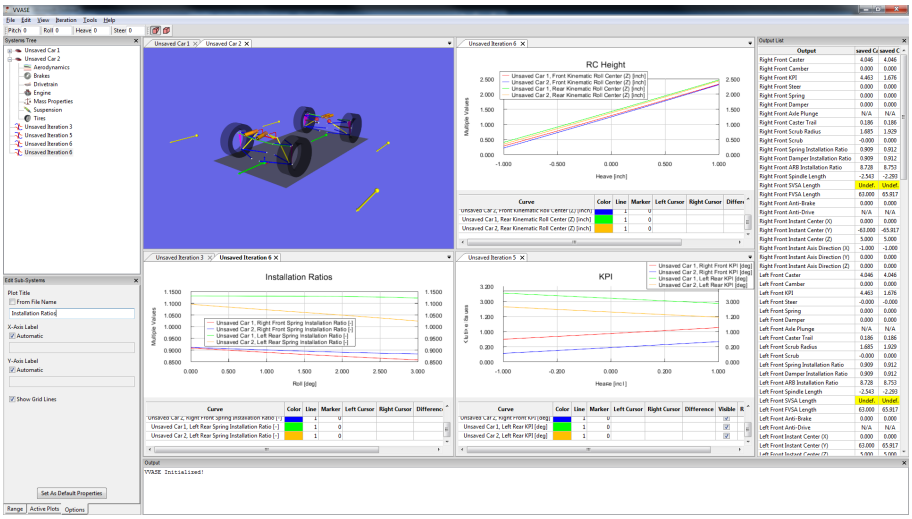

Figure 1.2: Custom layout demonstrating multiple plots displayed simultaneously

1.2.5 Main Display Area

The *Main Display Area* is a tabbed window and contains a tab for each open file. Each file type contains a unique display. Car files show a 3D image of the vehicle in the specified kinematic state (see § 2), iteration files show a 2D plot showing calculated kinematic outputs across a range of kinematic states (see § 3 and optimizations show a set of controls that allow configuration of the optimization parameters (see § 4).

The tabs in this area can be dragged, docked or floated to enable viewing the 3D model of the car adjacent to an iteration plot, or multiple iteration plots simultaneously. See Fig. 1.2 for an example.

1.2.6 Kinematics Toolbar

The *Kinematics Toolbar* is where the user can define a kinematic state for which the kinematic outpus shall be calculated. As the user types in the Pitch, Roll, Heave and Steer boxes, the kinematic outputs are computed for all open car files and the *Main Display Area* and *Output Lists* are updated. The units for these inputs are determined by the options set in the options dialog (see § 1.3.1).

1.2.7 3D Toolbar

The *3D Toolbar* only affects the *Main Display Area* for car files. There are two toggle buttons which switch between perspective and orthographic view

modes. Perspective mode draws objects in the foreground with a larger scale than objects in the background (the way objects appear in the real world). Orthographic mode draws all objects at the same scale. While perspective mode tends to be more pleasing, orthographic view can be helpful for example when visually determining how well points on opposite ends of the car line up.

1.2.8 Menu Bar

The application Menu Bar is fairly self explanatory. The *File* menu permits saving, loading and closing files. The *Edit* menu contains entries for undo/redo and cut/copy/paste. The *View* menu gives the user some control over the display of toolbars and the *Output Pane* contents. The *Tools* menu includes an item to display the *Options Dialog* (see § 1.3. The *Help* menu includes a link to open this document and can display information about VVASE.

Depending on what type of file is active, there may be an additional menu entry with specific options pertaining to the active file.

1.3 Options

In addition to display customization, there are several ways in which the behavior of VVASE can be adjusted to suit user preferences. All preferences are automatically saved when VVASE exits and loaded each time it starts.

1.3.1 Units

VVASE allows the user to specify which units to use for each type of unit. After selections are made and changes are applied, all inputs and outputs are updated to be consistent with the specified units. The options tab containing the units options is shown in Fig. 1.3.

1.3.2 Digits

If desired, the precision with which values are displayed can be adjusted. Optionally, value can also be displayed in scientific notation or while preserving a constant number of significant digits. The options tab containing the digits options is shown in Fig. 1.4.

1.3.3 Kinematics

When applying the specified kinematic state to the car, VVASE applies the rotations about the center of rotation specified here. The default location of $(0, 0, 0)$ is probably fine for applying roll motion, but when applying pitch, the default center of rotation is likely to result in excessive vertical displacement at the rear end. This can be resolved by moving the center of rotation backwards.

A frequently asked question is “why doesn’t the center of rotation match the roll center?” The answer is that cars do not roll about the roll center.

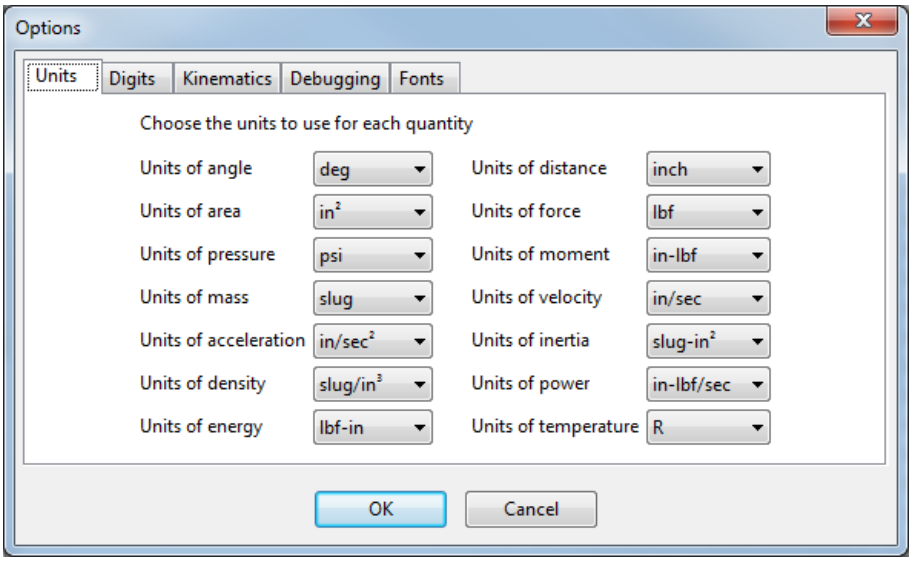

Figure 1.3: Units options

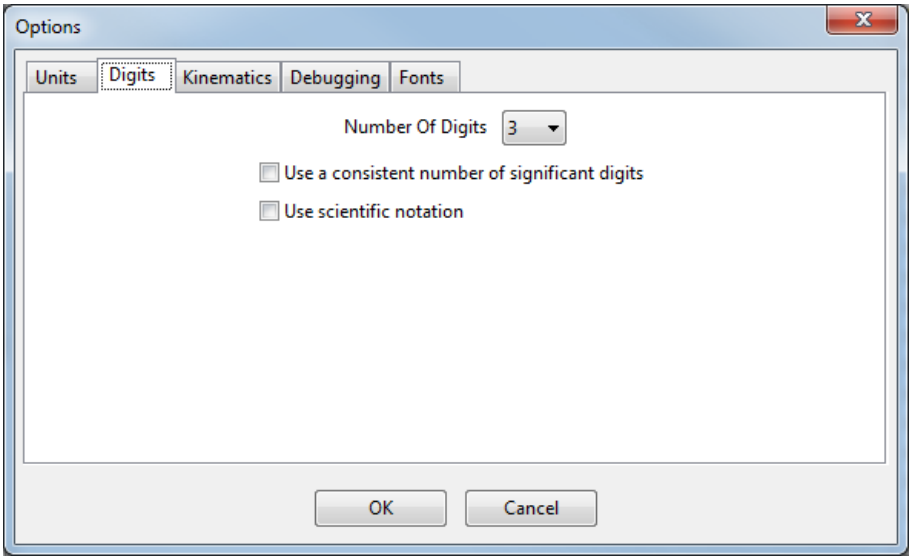

Figure 1.4: Digits options

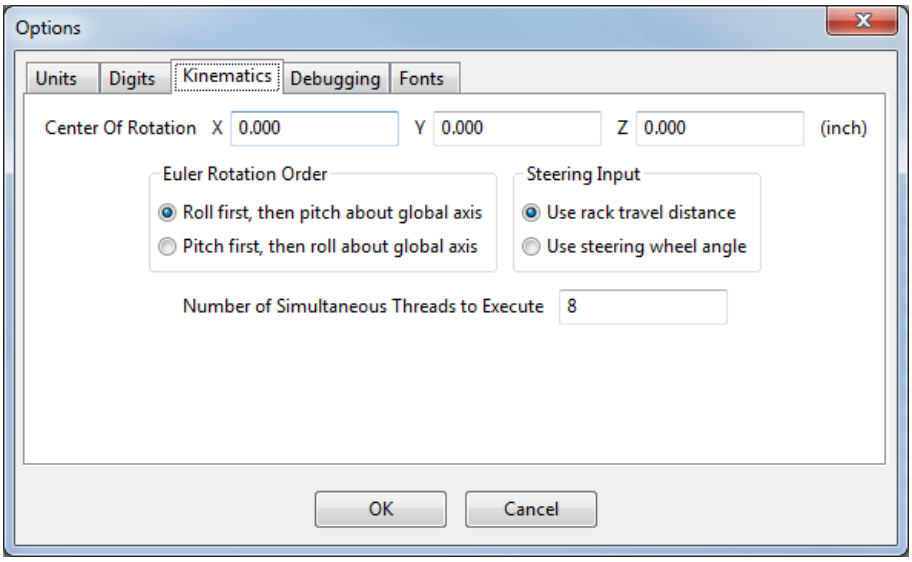

Figure 1.5: Kinematics options

Since tires and springs (and dynamically, dampers) are involved, the actual instantaneous location of the vehicle's rolling motion is the result of the forces acting on the components of the vehicle. Since the kinematics model does not include any forces, no attempt is made to compute the actual attitude that a car would reach during dynamic or even steady-state conditions. Instead, VVASE answers the question "if the car were to attain this attitude, what are the values of the relevant kinematic outputs?"

Additionally, the order in which the pitch and roll rotations are applied can be changed.

The "Steering Input" option allows the user to specify how the Steer input should be interpreted. In the case of "Use rack travel distance," the input is interpreted as a distance and applied directly to the locations of the inboard tie rods. In the case of "Use steering wheel angle," the input is interpreted as an angle, and converted to linear motion of the steering rack according to the specified steering ratio.

The number of threads to execute is generated by estimating the value that would make most efficient use of the available CPU cores. For large optimizations, it may be desirable to fine-tune this value in order to ensure the best possible performance.

The options tab containing the kinematics options is shown in Fig. 1.3.3.

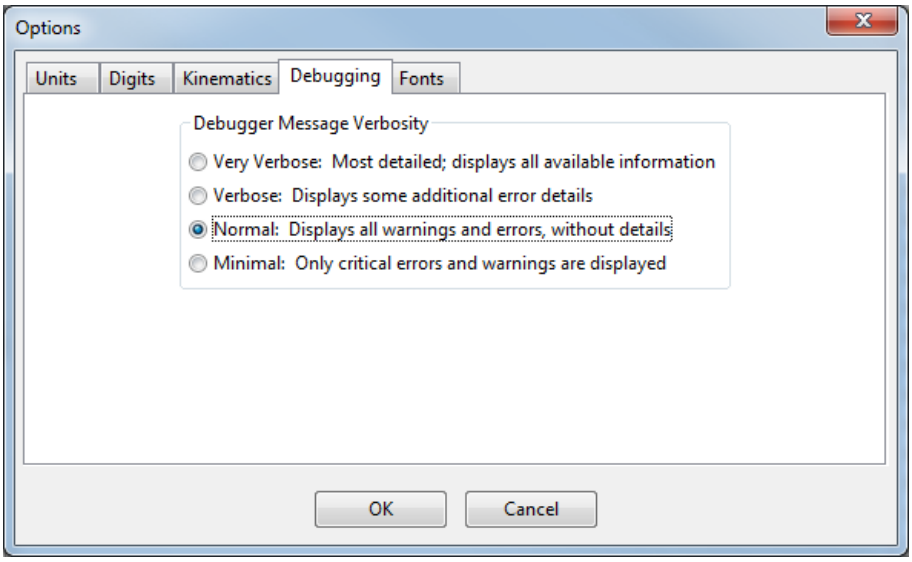

Figure 1.6: Debugging options

1.3.4 Debugging

The debugging options apply to the messages that appear in the *Output Pane*. Generally, the default level (Normal) is ideal, however in the event that a particular suspension is not properly solving, it can be helpful to make the debug level more verbose. The options tab containing the debugging options is shown in Fig. 1.6.

1.3.5 Fonts

If desired, the fonts used for rendering the iteration plots and the font used for printing to the Output Pane can be changed. For best results, it is recommended to only select TrueType fonts for use with the iteration plots. The options tab containing the fonts options is shown in Fig. 1.7.

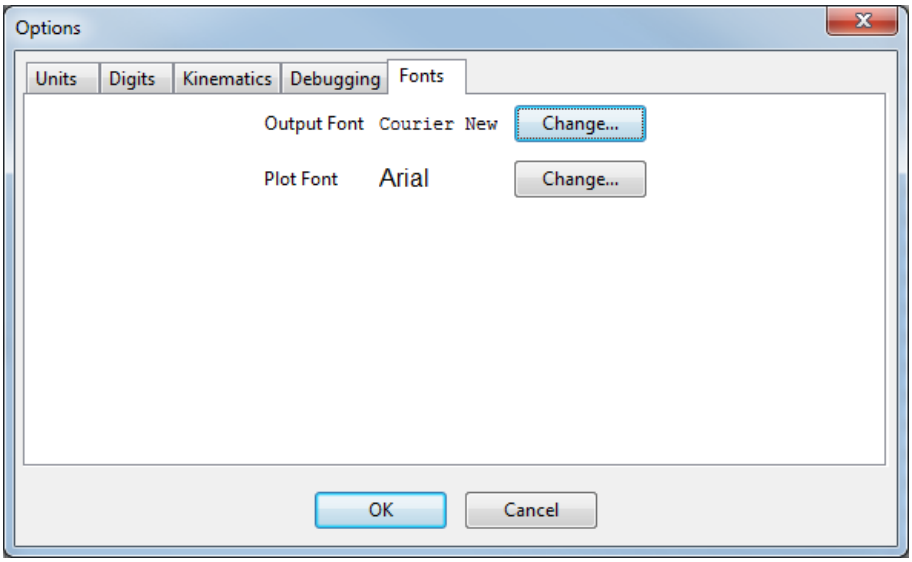

Figure 1.7: Fonts options

Chapter 2

Cars

Vehicles in VVASE are represented by .car files. These files store all information that describes the vehicle itself, including suspension hardpoints, mass properties, etc.

When a new vehicle is created, defaults values for all properties are created such that the suspension takes a reasonable form. This makes it easier to associate hardpoint names with the points shown on-screen.

When a car file is active, the *Main Display Area* contains a 3D rendering of the tires and suspension elements. This rendering is described in § 2.1.

Car files create several subsystems in the *Systems Tree*, with each subsystem having unique *Edit Panel* content. These subsystems are described in § 2.2.

2.1 3D Car Display

The 3D vehicle rendering shows the locations of all of the suspension elements and tires in the specified kinematic state. An example of the 3D rendering of a car is shown in Fig. 2.1. In the rendering, the green vectors represent roll centers and roll axes, and yellow vectors represent instant centers and instant axes.

It is possible to interact with the 3D display by clicking and dragging with the mouse to rotate, or using the mouse wheel to zoom.

By default, cars are rendered in perspective mode, but the *3D Toolbar* can be used to toggle between perspective and orthographic modes. See § 1.2.7 for more information.

The appearance of all of the suspension elements can be adjusted by clicking on the *Car* menu, and then selecting **Appearance Options**. Here, the color and size can be adjusted and the visibility of any object can be toggled on or off. Resolution, which can also be adjusted, corresponds to the number of triangles used to approximate round objects during the rendering process.

When a hardpoint is selected in the *Edit Panel*, a "helper orb" is drawn, which highlights the selected point in the 3D display. This can be helpful in understanding the naming of each hardpoint.

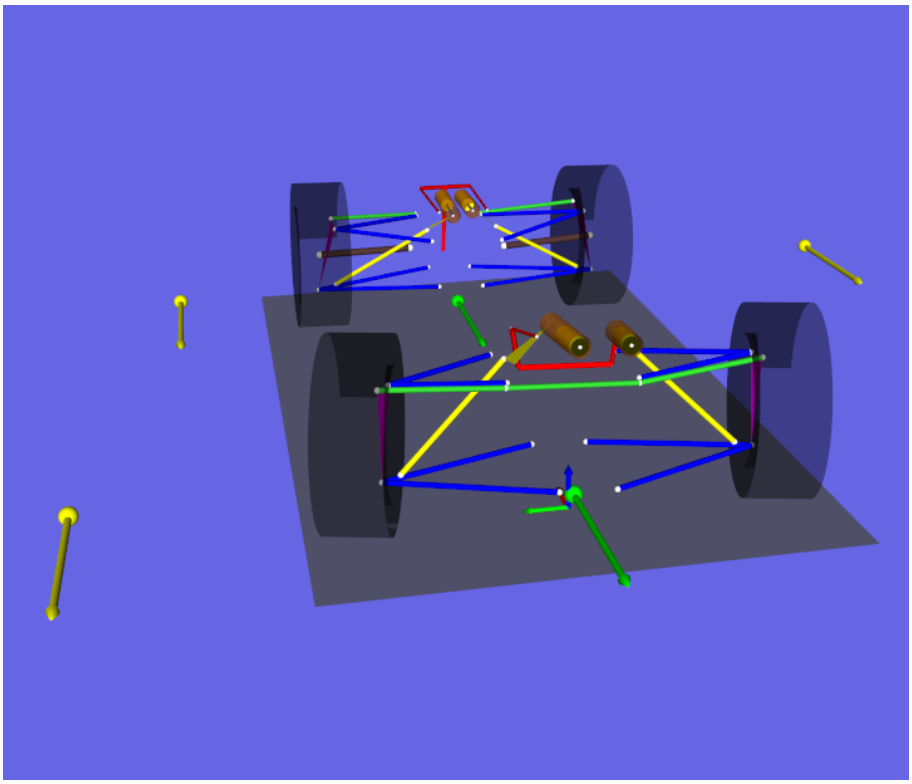

Figure 2.1: Example 3D car rendering for a kinematic state including roll and steer inputs

2.2 Subsystems

Each car file includes several subsystems. These subsystems can be accessed by expanding the active car in the Systems Tree. The subsystems are described below.

2.2.1 Aerodynamics

This subsystem is currently unused.

2.2.2 Brakes

The brakes subsystem includes three parameters that affect the calculated kinematic outputs. These parameters are toggles to indicate whether or not the front and rear brakes are inboard or outboard, and the portion of the braking effort that can be expected to come from the front tires. These parameters affect the calculation of the front anti-dive and rear anti-lift.

2.2.3 Drivetrain

This subsystem is currently unused.

2.2.4 Engine

This subsystem is currently unused.

2.2.5 Mass Properties

Although there are several mass properties parameters associated with each car file, only the z-component of the center of gravity is currently used. Currently it only affects the calculation of the anti-dive and anti-lift values.

2.2.6 Suspension

The suspension is the heart of the kinematics analysis. This is where all of the suspension hardpoints are defined, as well as a number of other suspension geometry settings. By default, the suspensions are assumed to be symmetric, but unchecking the corresponding option on the *Suspension* tab creates a separate additional tab for each corner instead of the *Front* and *Rear* tabs that exist by default. Also on the *Suspension* tab are options for the type of anti-roll bar present at each end (if any) and whether or not 3^{rd} springs are present at each end. Depending on the selected anti-roll bar and 3^{rd} spring options, a list of hardpoints may be displayed in the grid at the top of the panel. These hardpoints can be edited by typing in new values in this grid. Clicking on the hardpoint will highlight the corresponding point in the 3D display.

The additional tabs, whether there are two or four, are identical. Each tab corresponds to a corner or a pair of corners at one end of the car and includes a list of hardpoints, spring/damper options and static toe and camber settings. Just as with the *Suspension tab*, selecting a hardpoint in the list will highlight the corresponding hardpoint(s) in the 3D display.

By default, both ends of the car have pushrod suspensions. By changing the *Attachment* to the **Upper A-Arm** or a high location on the **Upright** and modifying the bellcrank hardpoints, pullrods can also be modeled. Outboard spring/damper units and rocker arms can be modeled by selecting the **Outboard/Rocker** *Actuation Type*.

Static toe and camber settings affect only the calculated steer and camber values.

2.2.7 Tires

The diameter and width of all four tires may be specified individually. The width is for display purposes only. The diameter affects the tire rendering as well as the kinematic calculations. It is assumed that the tires are thin, rigid disks. For all kinematic states the tires are assumed to rest on the ground.

Chapter 3

Iterations

Iteration objects provide a means of plotting kinematic outputs across a range of kinematic states. The settings for these plots are stored in .iteration files.

When a new iteration is created, default settings for the iteration are loaded from the configuration file. These defaults can be configured by the user (see § 3.4).

When an iteration file is active, the *Main Display Area* contains a rendering of the plot, as configured by the user. The user can interact with the plot and perform mathematical operations on plotted data. More information is in § 3.1 below.

Also when selected, the Edit Panel will contain three tabs. These tabs are described in § 3.2-3.4 below.

Behavior of the iteration can be further customized by using the options provided in the *Iteration* menu, which is described in § 3.5.

3.1 Plot Display

The plot is displayed in the *Main Display Area*. Below the plot is a list of all of the active plots. Here, curves can be toggled on/off or optionally plotted against the right y-axis instead of the left. The color, size and marker size for each curve can be independently adjusted. Right-clicking on the plot are or the plot list will generate a context-sensitive pop-up menu. These pop-up menus provide a means of performing mathematical operations on plotted data, fitting curves to plotted data, changing plot background colors, toggling the legend on or off and more. The legend can be dragged and dropped to place it in the plot area as desired. Double-clicking in the plot area introduces a cursor, which can be dragged across the extents of the plot area. The values corresponding to places where the cursor crosses the plot curves are displayed in the grid below the plot area.

An example of an iteration plot is shown in Fig. 3.1.

3.2 Range Tab

The *Range* tab describes the range of kinematic states which are to be represented on the plot. Any combination of pitch, roll, heave and steer can be used as start and end conditions. The number of points used on the plot can

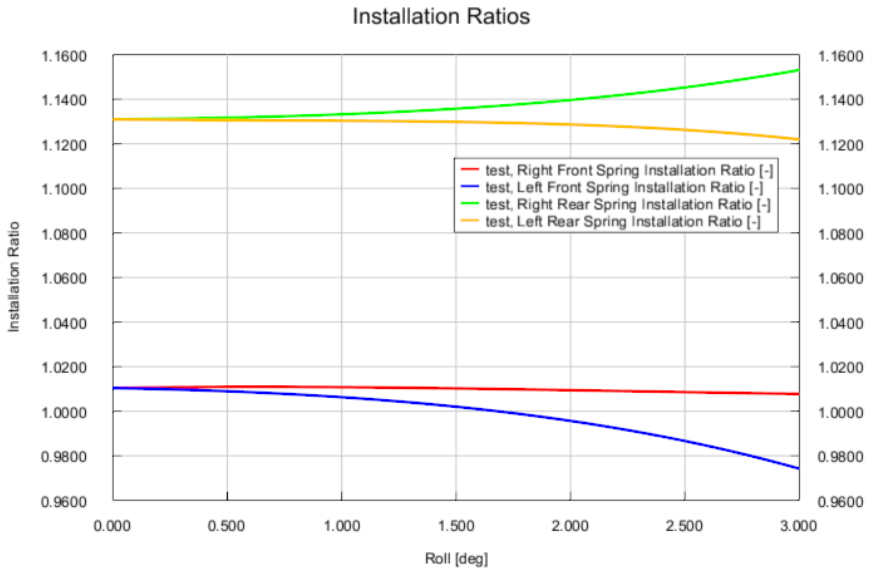

Figure 3.1: Example iteration Plot Display showing the installation ratios at all four corners of a vehicle while experiencing roll

also be adjusted. Generally, kinematic outputs change slowly and smoothly with respect to the kinematic states. For these cases, smooth curves can be produced with a relatively few number of points. Occasionally, however, an output will appear to change rapidly and more points are required to produce a smooth curve. It is advisable to use as few points as possible, especially if the iteration is to be associated with several open car files, as each additional point requires another sequence of calculations for each associated car.

3.3 Active Plots Tab

The *Active Plots* tab allows the user to specify which outputs should be plotted in the active iteration file. It is possible to select as many outputs for plotting as desired.

3.4 Options Tab

The *Options* tab allows the user to manually specify the plot title and axis labels. There is also an option that permits toggling the grid lines on and off.

At the bottom of the panel is a button labeled **Set As Default Properties**. Clicking this button will save all of the current plot configuration

options to the configuration file. Each time a new iteration file is created, this configuration will be applied.

3.5 Iteration Menu

All iteration data is exportable into a .csv format which can be opened with Microsoft Excel or other tools. This can be done by selecting the **Export Data** entry in the *Iteration* Menu.

By default, iterations auto-associate with all open car files. This means that the desired kinematic outputs are plotted for every open car. This makes side-by-side comparisons of different geometries easy. If this is not desired, the **Associated Cars** entry in the *Iteration* menu can be used to specify which cars should be analyzed by the iteration file.

VVASE will automatically select the correct independent variable for the plot when only a single kinematic input is varied. For cases when multiple kinematic inputs are varied, the user can select the independent variable by selecting the *Set X-Axis* entry in the *Iteration* menu.

Chapter 4

Optimizations

VVASE performs optimizations through the use of a genetic search algorithm. A brief description of genetic search algorithms is given in § 4.1.

When an optimization file is active, the *Main Display Area* contains a panel which allows complete control over an optimization. Fig. 4.1 shows the layout of this panel. From the top down, the panel includes controls for defining algorithm parameters (see § 4.2), a list of the variable inputs (see § 4.3) and optimization goals (see § 4.4). During an optimization, the progress bars at the bottom of the panel give an indication of how much time remains in the optimization process.

There is no *Edit Panel* content associated with optimization files.

4.1 Genetic Search Algorithms

Genetic search algorithms are a stochastic optimization, modeled on the process of evolution and natural selection. They work by applying a "survival of the fittest" approach to optimization problems. Generally, these algorithms must take the following steps:

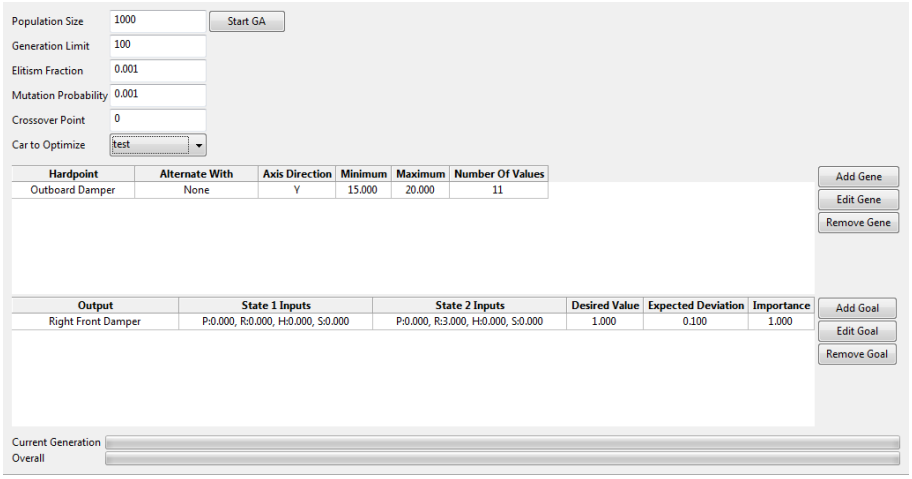

Figure 4.1: Genetic optimization panel

1. Randomly create initial population
2. Assign a fitness score to each population member
3. Sort the population according to the fitness score
4. Breed the members with the highest fitnesses to create the next population
5. Repeat steps 2 - 4 until reaching a predetermined fitness score, number of iterations or other termination criteria

To fit a problem into the required framework, each member must be representable by some type of code. With respect to the optimization, this code must describe the member entirely. This code is the genotype. Each code describes a unique representation of the population member. The representation corresponding to a genotype is a phenotype.

Additionally, a method is required for determining the fitness of each phenotype. This fitness score must be sortable.

4.2 Algorithm Parameters

The interface to the genetic search algorithm implemented in VVASE takes several top-level parameters as well as specific parameters for describing the genes and goals. This section defines the top-level parameters that control the process described in § 4.1 above. Setting the parameter values for specific optimizations takes some trial and error. The discussion below includes some recommendations and guidelines for setting the parameters, but ultimately the best choices for these parameters are optimization-dependent.

4.2.1 Population Size

The population size is the number of genotypes that are included in the simulation and scoring of each generation. The population size should be commensurate with the search space. When there are many possible genes, the population size should be larger.

4.2.2 Generation Limit

The generation limit sets the run length for the optimization. The generation limit should be sized proportionally to the number of goals included in the optimization. It is not uncommon for convergence rates to be slow for the early generations. If the population size is very large, it is also likely that more generations must be simulated in order to converge successfully.

4.2.3 Elitism Fraction

During the breeding process, it is possible that all of the offspring for a given generation fail to achieve a fitness equal to the best fitness from the previous generation. To guarantee that the best solutions are always carried from one generation to the next (and thus always part of the gene pool), an elitism fraction may be defined. When the elitism fraction is greater than zero, there will always be at least one genotype carried exactly as-is into the next generation. When used, it is best to keep the elitism fraction small, however, to ensure that the largest possible number of genotypes are available for the algorithm to use in finding better solutions.

4.2.4 Mutation Probability

The mutation probability describes the chance of a single gene being altered during the breeding process. When a gene is selected for mutation, it is randomly altered such that the result is still within the range of the original gene definition. Mutation can be used to help broaden the search space, but it tends to slow convergence rates, especially when a high mutation probability is used. If the mutation probability is increased too much, it is possible that the algorithm will fail to converge at all.

4.2.5 Crossover Point

The crossover point defines the point at which a genotype is split during the breeding process. For example, if there are four genes, a crossover point of two means the first two genes will come from one parent, where the last two will come from the other.

If the crossover point is set to zero, the selection of genes is randomized. There is still a guarantee that approximately 50% of the genes will come from each parent, however it is possible for every other gene to come from the opposte parent, that is, there are no "blocks" of genes coming from each parent in the way that a specified crossover point works.

4.2.6 Car To Optimize

This drop-down box is used to specify which of the cars the optimization applies. The original car is not modified; instead a new car is created at the end of the optimization, which represents the solution from the last generation having the best fitness score.

4.3 Genes

The genes define which portions of the car may be modified as part of the search. The parameters used to identify genes and the values they are allowed to assume are discussed below. Genes can be added or removed by clicking

the **Add Gene** and **Remove Gene** buttons to the right of the list of genes. Existing genes can be edited by doubling-clicking an entry in the list of genes or by clicking the **Edit Gene** button.

4.3.1 Hardpoint

The hardpoint parameter defines which of the suspension points is the phenotype for this gene. As the encoding of the gene changes, it is the location of this point that will be modified.

4.3.2 Alternate With

In some cases it is desirable to lock two hardpoints together such that that share a common coordinate value. This is sometimes useful for modifying inboard control arm positions, for example; both the forward and rearward points can be modified together.

This parameter is optional.

4.3.3 Axis Direction

This specifies which of the three ordinate dimensions to vary. If it is desired for a single hardpoint to be optimized in more than one direction, multiple genes must be defined, each with a different axis direction specified.

4.3.4 Minimum

This specifies the minimum allowable value for the phenotype to assume.

4.3.5 Maximum

This specifies the maximum allowable value for the phenotype to assume.

4.3.6 Number of Values

The number of values defines the granularity of the search space. The location of the hardpoint will only be able to assume certain finite numbers in the range from minimum to maximum. Using a large number of values can significantly increase the search space and may require increasing the population size in order to ensure convergence.

4.4 Goals

The goals define which kinematic outputs are targeted in the optimization. In addition, the fitness function is formulated based on these values. Goals can be added or removed by clicking the **Add Goal** and **Remove Goal** buttons to

the right of the list of goals. Existing goals can be edited by doubling-clicking an entry in the list of goals or by clicking the **Edit Goal** button.

4.4.1 Output

This specifies which kinematic output is targeted.

4.4.2 State 1 Inputs

The kinematic inputs are roll, pitch, heave and rack travel. This is the kinematic state at which the output is evaluated.

4.4.3 State 2 Inputs

Optionally, the user may check the option for **Optimize difference between states**. If selected, two text boxes are available for each kinematic input. The kinematic output will be evaluated at both conditions and the difference between the output at the defined states, rather than the value of the output itself, becomes the target of the optimization.

4.4.4 Desired Value/Desired Change

The desired value defines the criteria for the optimization. If the option for **Optimize difference between states** is selected, the criteria becomes the change between the output at the defined states, rather than the value of the output itself.

4.4.5 Expected Deviation

The expected deviation is the anticipated error between the achieved solution and the desired value or desired change. This is included as a weighting factor in the fitness function. It is useful when multiple goals are included in an optimization, and given the input parameters it is easier to achieve a closer fit for one output than another, or alternatively, when one goal has a large value and another has a small value. Without including the expected deviation, the fitness function may be dominated by a single goal, resulting in other goals being ignored during the optimization.

4.4.6 Relative Importance

The relative importance is a weighting factor included in the fitness function. Typically the importance values may be set to one, but if certain goals are not being achieved during optimizations, the relative importance can be used to assign more or less weight to certain goals.

4.5 Hints

There is no substitute for experience in creating optimization and observing the results achieved, but without some guidance it can be difficult to get started. In this section, some additional details on genetic search algorithms are discussed and how they might vary from traditional optimization techniques. There are also some best practices and guidelines that may be helpful when setting up optimization parameters.

4.5.1 Differences from Classical Optimization

Genetic search algorithms operate very differently from classical optimizations. This difference in operation principles is what provides the ability to tackle complex problems, but it can also lead to poor results if the differences are not well understood.

Classical optimizations have a value or a set of values (similar to genes) that are varied in order to attempt to minimize or maximize a fitness function (which could be the same as the fitness function defined by setting optimization goals). The values are usually altered according to the change in the resulting fitness score. For example, if the goal is to find the value of x that minimizes $f(x)$, for some algorithm the sequence might look something like this (assume an initial guess of $x = 0$):

$$\begin{gathered} x = 0; f(x) = 10 \\ x = 5; f(x) = 6 \\ x = 7; f(x) = 3 \\ x = 9; f(x) = 4 \\ x = 8; f(x) = 2 \end{gathered}$$

Initially, the algorithm took a relatively large step and guessed five in the second iteration. The function evaluates to a smaller value, so it keeps going in the direction of increasing x. This works for another iteration, but in the fourth iteration, $f(9)$ evaluates to four, which is greater than the result at $x = 7$. So the algorithm takes a step back and guesses eight, which must have been the right move, since $f(8)$ evaluates to two. Or maybe not — what if the global minimum for $f(x)$ occurs at $x = 20$? Apparently, this algorithm would never reach that point, instead ending the search at a local minimum in the vicinity of $x = 8$. This is a property of many classical optimization algorithms — they tend to converge at local inflection points instead of guaranteeing a global optimum.

Genetic search algorithms certainly do not guarantee arrival at a global optimum, but they can help to avoid this common trap. With a sufficiently large population size, genetic search algorithms can cover a broad portion of the solution space, which can provide confidence that a local optimum has been found. When the solution space has many local inflection points and/or

the solution space contains steep gradients, it is often helpful to increase the population size.

Another difference is the way in which the independent variables are represented. Classical algorithms store these values directly. Using the above example again, the algorithm tried $x = 7$ and then $x = 9$, and when the evaluation result increased, the algorithm then tried $x = 8$. A different algorithm might have tried $x = 7$, $x = 9.5$ and then $x = 8.2$. All of these values of x are real numbers, and therefore they are valid inputs.

Genetic search algorithms work differently — the allowable input values are defined when the optimization is started. The algorithm must be told that x is allowed to take on the values of seven, eight and nine. Or, it is possible that the algorithm was not told it was allowed to choose $x = 8$ and it could have returned $x = 7$ as the optimal solution.

This might sound undesirable, but it can be a very useful property. By limiting the resolution of the solution space, the number of possible solutions is reduced from infinity to a finite number. For solution spaces with smooth changes and a global optimum that is very different from the local optima, larger steps can be used to reduce the computational effort required to converge. It is an important point, however, since it behaves differently from other algorithms and can lead to unexpected results if it is not considered.

These differences make genetic search algorithms well-suited to solving optimization problems with complex, high-dimensional solution spaces. These differences also make genetic search algorithms comparatively inefficient for simple optimizations. In the event that a simple optimization is required, it is beneficial to include only genes that directly affect the goals, use a large number of values that these genes may assume, choose a relatively large population size and choose a low generation limit.

4.5.2 Best Practices

This section summarizes some genetic optimization best practices.

Test for Proper Population Size Too small and the solution may be suboptimal, too large and CPU cycles are wasted. A helpful test is to set an artificially low generation limit (maybe just a few generations) and run the optimization several times. Examine the optimization result. Do all of the genes have the same phenotype, or is there variation in the results? If the results are all very similar, it is likely that the solution space is too small for the specified population. After just a handful of generations, there should be a great deal of variation (for a properly sized population, the phenotypes should be random at this point). Knowing if the population is too small is more difficult and often required running a full optimization and checking for convergence.

Using Enough Generations For complex optimizations, be sure to use a generation limit that is high enough to allow the solution to converge. As

a rough starting point, consider using the following expression: $limit = 10^{numberofgoals}$.

Start coarse and refine To reduce computational effort and improve solution speed, start with relatively small numbers of values for each gene. After narrowing the solution space by running this coarse simulation, update the allowable range of each gene to be smaller, centered around the most recent optimum, keeping the number of values constant. Of course, if the solution space has steep gradients, it may be required to start with a large, finely distributed solution space from the beginning.

Separate independent optimizations Genetic search algorithms are helpful for finding solutions to optimization problems that are tightly coupled — when every gene has some influence over every goal. If it is desired to optimize two independent goals (for example, front roll center movement and rear roll center movement), it is best to use separate optimizations.