If at first you don't succeed, you probably didn't run enough simulations.

— John R. Vacca, November 1995 Byte magazine

6. The Simulation

Using a multibody passive-legged class of vehicle as a solution to the all-terrain mobility problem presents a vast design space of particular implementations. Specifically, there is a multitude of possible crawling vehicle configurations, and for each of these different robot designs, there is a virtually limitless number of possible motion programs that could be used. Note that a motion program or gait that works well with one robot design often does not work well with another design. Solving the all-terrain mobility problem requires the compatible operation of both a good robot design and good motion program designs.

Sifting through such vast design possibilities for the best solution is not a job for pocket calculators and drawing boards—more powerful tools are required to obtain good, timely results. As with other complex engineering design efforts, simulation is often the tool of choice because it enables the testing of design alternatives without having to physically construct them.

Having presented the decision-making process for the conceptual design stage in Chapters 2 and 3, the simulation tools described in this chapter focus on the configuration design stage. As explained in Section 1.9.2, tools that incorporate synthesis functions and qualitative analysis at the configuration design stage can enable designers to search the design alternatives more rapidly and at less cost than

tools that require the detailed specification of proposed design alternatives as inputs. This is because configuration designs are less expensive to develop and simpler to analyze than full, detailed designs, and because costly redesigns at the detailed design stage can be avoided if the best configuration design is first found via a thorough search of the configuration design space.

The engineering groundwork for the simulation has been laid in the previous two chapters. Specifically, the geometric parameters described in Section 4.1 enable a multitude of possible robot designs to be modeled. Furthermore, the modeling techniques developed in Ch. 4 can process and analyze any arbitrary sequence of leg pair position (x, y, z, roll, pitch, yaw) datasets for the crawling vehicle. And finally, the motion programming techniques presented in Ch. 5 provide that position data as a function of time for every leg pair of the crawling vehicle. Hence, all the prerequisite pieces are available to assemble into simulation tools for enabling designers to test and compare a large variety of possible crawling vehicle configurations and their locomotion methods.

Thus, the simulation code developed as part of this research consists of algorithms and data structures that embody the modeling, synthesis, analysis, and trajectory specification techniques described in Ch. 4 and Ch. 5, together with a user interface and code for displaying renderings of robot motion. These have been encoded into a C language computer program.

Figure 6.1 shows the main software modules of the simulation tools, along with their inputs and outputs.

The following sections explain each of these software modules and their functions.

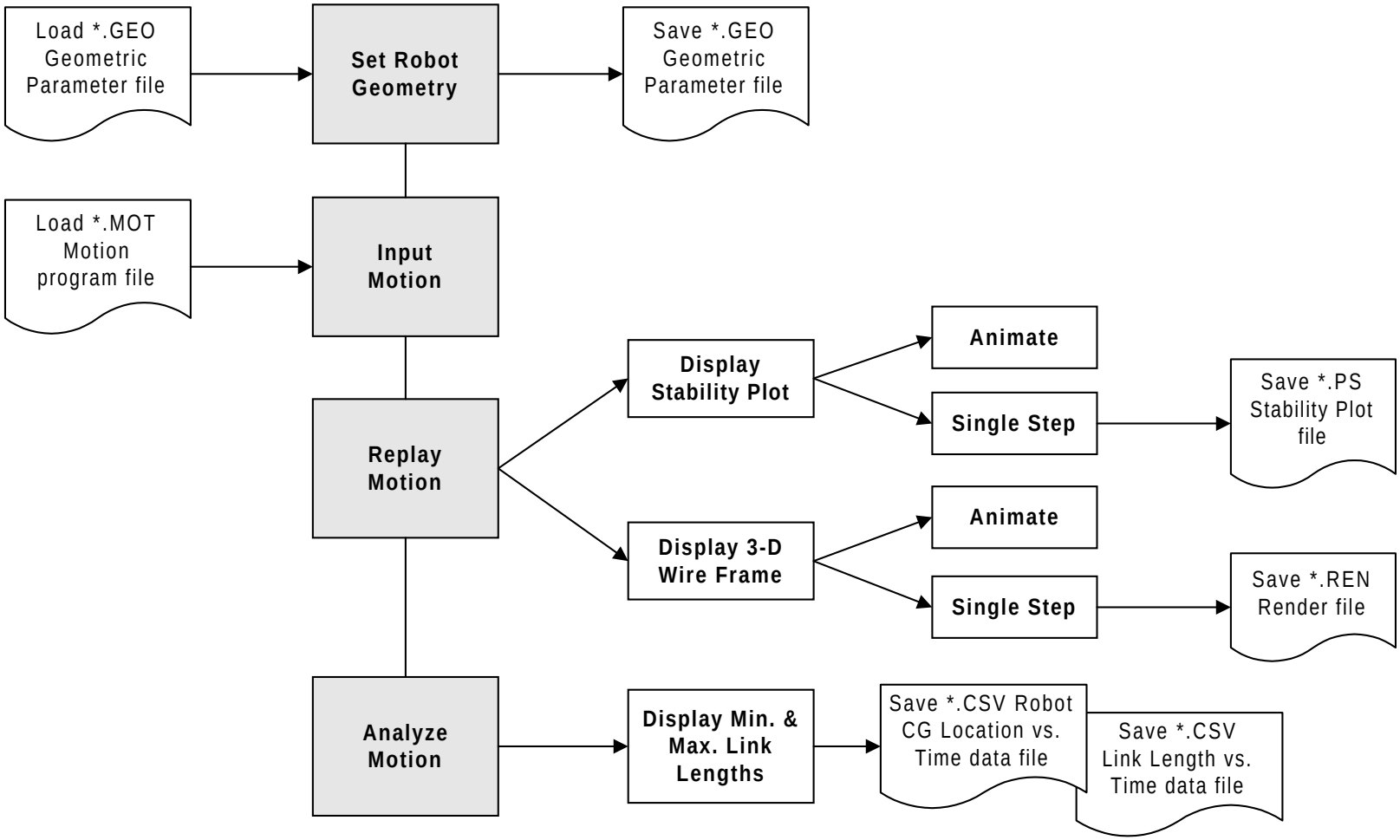

Figure 6.1 – Crawling Vehicle Simulation Program Main Modules, with Inputs and Outputs

6.1 The Set Robot Geometry Module

The Set Robot Geometry Module implements the geometric parameters for defining crawling vehicle designs explained in Section 4.1. Using the program, the 31 geometric parameters that define a crawling vehicle design can be either entered via a keyboard, loaded from any user-specified file with the extension ".GEO", or loaded from a "default.geo" file. The geometric parameters can also be edited via keyboard and saved to a user-specified file.

Once entered, the geometric parameters are stored in a special data structure that describes the crawling vehicle geometry. This data structure is used by all the other software modules of the simulation.

6.2 The Input Motion Module

The Input Motion module loads motion program files, which are designated by the ".MOT" filename extension, and stores them in RAM memory. These files use the syntax described below.

"Motion programming" technology embodies both the geometrical information used to define motion trajectories and the programming aspects of creating an interpretable language to describe and sequence the motion, so that it can be interpreted and executed by a machine.

Recall from Section 5.1 that a motion program for a leg pair trajectory segment consists of path information (two parametric cubic coefficient matrices) and timing information (the start and stop times for the trajectory segment curve). One or more of these trajectory segments can be used to define a leg step for a leg pair. At this point, it is not clear whether absolute or relative times should be used.

The overall motion program file is made up of a series of program blocks that each specifies one trajectory segment for a particular leg pair. The motion program block for an individual trajectory segment consists of the parametric cubic coefficient matrices for the *xyz* and $\gamma\beta\alpha$ curves, the time at which the leg pair is to start following the parametric cubic (PC) curves, and the time at which it is to complete tracing the PC curve path. Specifically, the syntax for each program block is:

- a numeral indicating which leg pair to program
- the start time
- the stop time
- the *xyz* coefficient matrix (4 rows of 3 numbers)
- the $\gamma\beta\alpha$ coefficient matrix (4 rows of 3 numbers)

As discussed in Sections 5.2.4 and 5.3.4, the start and stop times for the leg steps are determined at the onset of each new wave, based upon the currently chosen wave interval, step interval, and transfer phase time values. While the simulation currently uses absolute times, times expressed relative to their immediately preceeding events could also be used.

The motion program interpretation portion of the simulation code requires that the program blocks for a particular leg pair be listed in the motion program in same sequential order as they are to be executed. Thus the program block for the second stride of a leg pair must be written in the motion program below the block for the first stride. However, it does not matter if leg pair 4 has its second stride listed before the first stride of leg pair 1.

This method makes it possible to program several leg pairs to move simultaneously. Note that there is no guarantee that the robot will remain stable. However, as will be

discussed in the next section, the simulation model can detect when the robot is in danger of falling over.

6.3 The Replay Motion Module

The Replay Motion module interprets the motion program stored in memory, simulating the motion to determine the position of each leg pair as a function of time, and thus, acting out the motions that would be produced if the motion program were being executed by a crawling vehicle of the stored design.

The resulting outputs can be displayed as animations of either stability plots or 3-D wire frame images of the robot that are viewable from 6 different angles. If desired, the designer can replay these animations in single-steps for closer scrutiny. In the case of the stability plots, the numerical value of the stability margin is also displayed for each time increment. Any individual stability plot can be output as a PostScript™ file (*.PS) suitable for printing (for example, Fig. 4.8). A series of wire frame images can be output to a render file (*.REN), which can later be read by a custom written program running on a Silicon Graphics workstation. This program, written by the author in C-language with IRIX-GL extensions, reads the render file, creates a 3-D surface model of the robot for each individual step of the motion program, renders the model (i.e. adds color, removes hidden lines, lights it, shades it, and adjusts it for viewing angle and perspective), and then displays it on-screen. The render program also enables the user to save these images to disk as a sequentially numbered series of files. These image files can later be combined into animations using commercially available programs or can be used individually as stills, such as the one shown in Fig. 3.10.

6.4 The Analyze Motion Module

Like the Replay Motion module, the Analyze Motion module also interprets the stored motion program to determine the position of each leg pair as a function of time. But in this case, the emphasis is not on replaying the bulk motion produced by the motion program, but rather on extracting the details of the motion in numeric form. Specifically, for the given motion program, the Analyze Motion module computes both the position of the robot's center of gravity and its inverse kinematics at a user-defined time interval.

6.4.1 Robot Center of Gravity

At this, the configuration design stage, the center of gravity calculation is, of necessity, approximate. This is because the masses and mass distributions of both the leg pairs and the linear actuators are unknown. However, the overall robot center of mass can be approximated by using the centroids of the leg pair assemblies and actuation units, and by making reasonable assumptions about their relative masses and mass distributions.

Specifically, it is assumed that the mass distribution within each of the leg pairs is uniform and that it can be considered to be concentrated at the centroid of its payload box. All the interior leg pairs are assumed to have the same mass, while the front and rear leg pairs are assumed to have masses that are half that of an interior leg pair. Furthermore, the prismatic links and the spheric joints that make up the actuation units are considered to be massless.

This last assumption is based upon the following logic. Each group of 6 linear actuators belonging to the same Stewart-Gough platform mechanism can be lumped together as one mass. Specifically, because the Stewart-Gough platforms will be

designed to provide symmetric workspaces, the center of gravity of each will always be near to the midpoint between the leg pairs to which it is attached, *regardless of robot posture*. Therefore, it is possible to divide the mass of each Stewart-Gough platform and lump half of the mass with the leg pair behind it and the other half with the leg pair in front of it without introducing large errors. Having allocated their masses to the leg pair assemblies, we can now consider the Stewart-Gough platforms (and their component prismatic links) to be massless. Thus, this assumption is reasonable and, along with the other assumptions, yields answers that are sufficiently accurate for configuration design decision-making.

The Analyze Motion module outputs robot center of gravity data as a Comma Separated Variable (*.CSV) file that lists the time and the corresponding *x*, *y*, and *z* ordinates in columnar form. This format is easily readable by spreadsheet programs, enabling further analysis and graphing. Such a graph was presented in Fig. 5.2.

6.4.2 Synthesizing Link Lengths

In addition to computing the time history of the robot center of gravity, the Analyze Motion module also performs inverse kinematics. The inverse kinematics computation (described in Section 4.3) effectively synthesizes the lengths of the prismatic links that are required to realize the given motion program. At the end of a run, Analyze Motion displays the minimum and maximum link lengths for a given robot geometry and motion program, and the ratio of the minimum and maximum for each corresponding prismatic link type.

The link length versus time data for all the prismatic links is written as a separate Comma Separated Variable (*.CSV) file for each actuation unit of the crawling vehicle being analyzed. This format is easily readable by spreadsheet programs, enabling further analysis and graphing. The most common further analysis entails performing the numerical derivatives to compute the velocities and accelerations of

the linear actuators as functions of time, as described in Section 4.6. An example of the graphs that result from a link length data file and its subsequent spreadsheet analysis was shown in Fig. 4.10.

6.4.3 Evaluating Link Lengths and Assembly Checking

Assembly checking is the process of verifying whether a mechanism made up of links of given dimensions and with given joint angles and/or translations can be physically assembled or "put together". This is a crucial step in confirming the viability of any mechanism design. For the case of the crawling vehicle, it is important to verify that the prismatic links of the Stewart-Gough platform mechanisms can actually assemble throughout all the positions they will be required to assume during the execution of the motion programs specified for use with the vehicle.

There are two ways to approach this problem: the traditional "analysis approach" and its inverse, a "synthesis approach".

In the case of the crawling vehicle, the more traditional "analysis approach" would proceed as follows:

1. Specify the design of the entire vehicle physical structure, including the valid minimum and maximum lengths of all the prismatic joints (i.e. the linear actuators).

2. Specify all the desired motion the robot is to perform.

3. Simulate running the motion on the robot, and analyze the run by computing the inverse kinematics at many time steps.

4. Evaluate the test. Specifically, if any of the link length values are ever outside of the given valid range of length values, then there is a failure to assemble

and the given robot design cannot actually perform the given motion program.

5. Upon evaluation, if there are any assembly problems or if the design is suboptimal, go back to Steps 1 and/or 2 and redesign the robot, the motion program, or both, and try again.

In contrast, the "synthesis approach" procedure is as follows:

1. Specify the design of *most* of the vehicle physical structure, including everything *except* the link lengths of the prismatic joints.

2. Specify all the desired motion the robot is to perform.

3. Simulate running the motion on the robot and analyze the run by computing the inverse kinematics at many time steps.

4. Use the results of the inverse kinematics to synthesize the lengths of all the prismatic links needed to accomplish the given motion. Specifically, the minimum and maximum lengths of each of the prismatic joints are determined by searching the results of the simulation run analysis for the minimum and maximum lengths that occurred for each prismatic joint as the robot executed the motion program.

5. Evaluate the test. Specifically, if any of the synthesized links violate the design rules (to be discussed later in this section), then the given robot design cannot be made to be compatible with the given motion program with *any* possible set of prismatic link dimensions that conform to the design rules.

6. Upon evaluation, if the given robot design is shown to be unable to execute the given motion program, or if the design is suboptimal, go back to Steps 1

and/or 2 and redesign the robot structure, the motion program, or both, and try again.

Thus, while there is still a design, test, and evaluate cycle with the synthesis method, critical parts of the design, specifically the lengths of all the actuators, are determined automatically for the designer.

In certain cases, the synthesis approach can empower designers to make more informed decisions. If a motion program causes a robot design to fail to assemble, with the analysis approach there is no guidance on the cause of failure, and a designer might falsely assume that only the link lengths of the actuation unit need to be changed. Thus, with the analysis approach, designers might needlessly waste time and money running hopeless simulation tests. In contrast, with the synthesis approach, the designer can learn in a *single* simulation run that a particular robot design/motion program combination will not work with ANY possible link lengths of the actuation unit design, that the problem lies elsewhere in the system design, and that the basic robot design, neutral stance spacing, and/or the motion program must be changed.

Because of this advantage and because, at this point, we are still designing the vehicle configuration, it seems appropriate to use the synthesis approach for the simulation. What is ultimately desired is a robot capable of performing motions that will enable it to traverse the terrain; therefore, synthesis seems to be a more direct way of finding the required link dimensions to acomplish this. Later, when an actual robot is built, it will probably be desirable to include assembly checking code using the analysis approach in the control system.

Having chosen the synthesis approach, the assembly checking process is governed, not by comparisons to preset maximum and minimum values of the link lengths (as it would be if the analysis approach were used), but rather, by a set of rules that determine whether the synthesized link lengths violate certain design constraints or

are simply not feasible to construct. These design constraints are based upon the desirable properties for the actuation unit workspaces that were discussed in Section 2.2.1.1. Because the simulation uses a synthesis approach, the motion programs effectively define the actuation units. Therefore, in order to produce the desired workspace characteristics, certain constraints should be applied to essentially all motion programs. These constraints include an assembly constraint, a workspace uniformity constraint, and a workspace lateral symmetry constraint.

6.4.3.1 The Assembly Constraint

The assembly constraint ensures that the programmed motions do not exceed the workspace of the robot. Because of the synthesis basis of the simulation, it does not simply say, "you can't do that" when a motion goes too far. Instead, the Analyze Motion module of the simulation code returns the link lengths as they are synthesized from the given motions, and outputs the ratios of their minimum to maximum lengths to the computer display. It is then up to the designer to determine whether these lengths ever become too short or too long during the given motion. For the test cases presented later in this dissertation (Ch. 7), the rule used to determine whether the assembly constraint is being obeyed is that the minimum length of each prismatic link must never be less than 60% of its maximum length during the performance of the entire motion program. This 60% ratio is intended to account for the workspace of simple linear actuators made of two sections, with one section sliding inside the other (see Fig. 3.8).

6.4.3.2 The Workspace Uniformity Constraint

The workspace uniformity constraint means that all the actuation units of the robot must have the same workspace. As explained in the Conceptual Design chapter (Ch. 2), this is a very desirable property that greatly simplifies both the construction and motion programming of such a vehicle. A rule to enforce this constraint is the same

60% rule as the assembly constraint, except that the minimum and maximum lengths used in the ratio are the minimum and maximum lengths of the equivalent prismatic links in ALL of the actuation units of the robot.

To identify the equivalent prismatic links, first recall that all of the actuation units of a crawling vehicle are of identical design. Then, by looking at the Stewart-Gough platform mechanisms from posterior to anterior, the 6 prismatic links of an actuation unit can be uniquely identified by such names as "top-left" or "bottom-right". Hence, to apply the workspace uniformity constraint to a robot with, say, 4 actuation units, the minimum link length determined for the top-left type actuator will be the minimum length found when comparing the top-left links of all 4 actuation units. Likewise, the maximum link length for the top-left type actuator will be the maximum length found when comparing the top-left links of all 4 actuation units. The same sort of comparisons will also be performed for the remaining 5 types of prismatic links.

6.4.3.3 The Workspace Lateral Symmetry Constraint

The workspace lateral symmetry constraint ensures that the workspaces of the actuation units enable the robot to turn left or right with equal facility—another desirable property mentioned in Ch. 2. A rule to enforce this constraint is the same rule as the workspace uniformity constraint, except that, by equivalent prismatic links, we now mean not just, for example, all the top-left links of all the actuation units, but rather that all of the top-left **and** top-right links collectively must be capable of the same minimum and maximum lengths. This effectively reduces the number of basic link types from six to three.

6.4.4 Using the Linear Actuator Workspace Constraints

Based upon these three constraints, the Analyze Motion module outputs the minimum, maximum, and the ratio of the minimum to maximum link lengths for

each of the three distinct prismatic link types (which can generally be uniquely identified as: top, middle, and bottom). Using these output values, the designer can see whether executing the given motion program on the given robot design is feasible using the chosen type of linear actuators (e.g. simple or compound). If the linear actuator dimensions are not feasible, the synthesis basis of the simulation analysis allows the designer to know that, in order to make the system work with the selected type of linear actuator, the redesign requires something *other than* simply changing the dimensions of the actuators.

6.5 Chapter Summary

The configuration design software tools created as part of this research simulate and analyze user-specified crawling vehicle designs and motion programs. These programs enable a virtually limitless number of crawling vehicles to be tested by using the parametric geometry specification described in Ch. 4. Similarly, they permit the testing of a myriad of possible motion programs based upon the methods described in Ch. 5.

The simulation supports configuration design decisions in a very goal-oriented sense. Instead of making a design and then simulating it to determine how much the vehicle can do, this simulation uses the approach of specifying what the desired motion capabilities are, and then deriving the actuator lengths and motions required to achieve that goal.

The underlying method used by the simulation is to interpret the motion program being tested so as to define the position of the robot at user-defined, evenly spaced, discrete times during its motion. Using this information, the program performs a quasi-static analysis of the motion at each discrete time step, as well as deriving motion analysis data, graphs, and animations that encompass the entire motion

program run. Specifically, each discrete position of the robot can be analyzed to determine the center of gravity, stability margin, and lengths of all the prismatic links required to obtain that position, as well as to display stability plots and wireframe renderings. At the end of a motion program run, the simulation analyzes it to display the maximum length, minimum length, and ratio of the minimum to the maximum for each distinct type of prismatic link. This enables the designer to determine whether the computed link workspaces are actually feasible. In addition, the simulation produces output files, including: geometric parameter files for defining robot designs, center of gravity versus time data files, prismatic link length versus time data files for all of the robot's actuation units, stability plot print files, and render data files. The center of gravity and prismatic link length versus time files can later be loaded into spreadsheet programs for further analysis and graphing. Similarly, the render data files can be read by a custom written program to produce both static and animated shaded images of the crawling vehicle performing locomotion.

These capabilities of the simulation modules make them valuable tools for aiding in both the hardware design of the robot’s configuration and the software design of its motion programs. In addition, the simulation code helps software design in the sense that many of its functions, including modeling, analysis, and motion program interpretation and execution, can also be used as part of the control software of an actual crawling vehicle.