Space Robotics Assembly Team Simulation

NASA P.O. No. H-34940D

Final System Integration Document
DCI-TR-041802-1

Submited by:

Mr. Mike Craft
Dr. Tom Howsman

Dynamic Concepts, Inc.

April 18, 2002

Table of Contents

1.0 Introduction 1

2.0 Software for the Evolutionary Design of Robotic Structures 2

3.0 Darwin2K Overview 2

4.0 Design and Simulation using Darwin2K 3

5.0 Darwin2K Modifications 5

- 5.1 Links 5
- 5.2 Joints 6
- 5.3 Mechanisms and Evaluator 6
- 5.4 Controller 8

6.0 Simulation and Evaluation using Darwin2K 8

- 6.1 Configuration Display (diplayCfgGL) 8
- 6.2 Simulation (evStandaloneGL) 9
- 6.3 Population Manager (pmStandalone) 10

7.0 Results 11

8.0 Summary 13

Appendix A – Darwin2K Source Code Modificatins 14

Appendix B – Darwin2K / UmbrellaBot Input Files 39

1.0 Introduction

The Space Robotics Assembly Team Simulation (Space Rats) is an expansive concept that will hopefully lead to a space flight demonstration of a robotic team cooperatively assembling a system from its constitutive parts. A primary objective of the Space RATS project is to develop a generalized evolutionary design approach for three classes of robots. These classes are inspection, transport, and assembly. In 2001, the Space RATS team met with potential customers in Marshall Space Flight Center's (MSFC) Engineering Directorate (ED), Flight Project (FD), Science Directorate (SD), and Space Shuttle Main Engine (SSME) program office. During these discussions the Space RATS team identified potential evolutionary design activities for each robot class. For instance, miniature robots could potentially be used to search for nicks and remove debris from the engine – tasks which are appropriate for an inspection and transport class robot. Discussions with SD have led to a concept for a space based manufacturing robot, which is an example of an inspection and assembly class robot. A meeting with the Advanced Projects Office (APO) in FD led to robotic construction team concepts that span all three classes.

The portion of the overall Space Rats program associated with the evolution of the robot morphology is the subject of this current contractual effort between NASA and Dynamic Concepts. The previous Support Equipment Evolutionary Design System (SEEDS) project was focused upon the evolution, via genetic algorithm, of an artificial neural network that was used to recognize certain defects in a surface. Likewise, Dynamic Concepts is currently researching a cooperative robotic construction methodology (stigmergy) under the NASA SBIR program. Each of these research activities plays a role in the overall Space Rats program, and each research area is complimentary to one another.

Work began on the SpaceRats morphology evolution project during the month of November, 2001, and concluded in April, 2002. The vast majority of the effort has been applied to the use and modification of Darwin2K, a robotic design software package, to analyze the design of a tube crawling robot. This robot is designed for carrying out inspection duties in relatively inaccessible locations within an engine similar to the SSME. A preliminary design of the tube crawler robot has been completed, and the mechanical dynamics of the system have been simulated. An evolutionary approach to optimizing a few parameters of the system has

been utilized, resulting in a more optimum design. The details of the Darwin2K software, and the tube crawling robot design are presented in the following sections.

2.0 Software for the Evolutionary Design of Robotic Hardware

An internet based investigation into the availability of software for the design of robotic morphologies was conducted early in the contract period of performance. While there are several research centers active in the application of evolutionary strategies to the design of robot morphology (e.g, Brandeis University, University of Sussex, JPL, etc.) there doesn't seem to be much publicly available software at this point. The available software consists primarily of general purpose evolutionary algorithms, and typically lacks the important specializations and underlying physics models required for robotic morphology design.

A notable exception to the general purpose software is the Darwin2K software package. Originally authored by Dr. Chris Leger as part of his Ph.D. program at Carnegie Mellon University, development of Darwin2K is now being performed by a small number of researchers and is hosted on the open source development site SourceForge.net. DCI has decided to utilize the Darwin2K software and has installed the software on several of its workstation Linux PC's. The installation of the software is not a trivial matter, and requires some familiarity with Unix, OpenGL, Xforms, and Perl. A set of notes detailing the installation of Linux 7.2 and Darwin2K within a typical MS Windows network was included in the Initial Process Document (report No. DCI-TR-012202-1).

3.0 Darwin2K Overview

Darwin2K is an open source software suite of tools for robot simulation and design optimization. Darwin2K allows a user to define a robot using a simple tree structure, draw from (or add to) a library of pre-defined robotic joints, define the robot's terrain and performance metrics, and simulate the kinematics and dynamics of the robot. An evolutionary design algorithm (genetic algorithm) is implemented to allow the user to optimize the robot kinematics, dynamics, components, and controller parameters. Darwin2K is a series of object-oriented applications created in C++ and is intended to be compiled and executed on Unix-based operating systems, such as Irix or Linux. It should be noted that Darwin2K is not an "end user" software product. In order to implement any type of complex robotic design, the user is required

to either modify or add to the Darwin2K source code. Further information about Darwin2K may be found at www.darwin2k.com.

4.0 Design and Simulation using Darwin2K

A significant effort has been made towards using the Darwin2K software suite to develop the design of a robot capable of crawling through a tube independent of the direction of gravity. In order to quickly develop a prototype robot, an existing robot design was modified to allow for the robot's legs to simultaneously make contact with an upper and lower surface. Diagrams of this robot are shown in Figures 1 and 2.

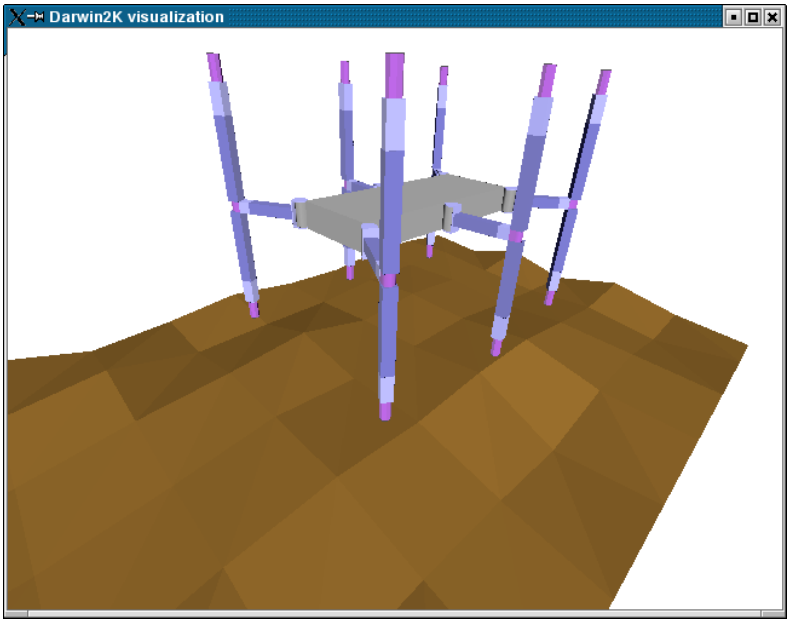

Figure 1 – Modified Darwin2K HexBot on Random Terrain

While this design appears to meet the mission requirements, it results in a complex configuration with 24 separate joints. In order for this robot to be effective, a controller would have to be developed to coordinate the complex motion of the robot's joints.

A different, less complicated design of a robot to perform the mission has been pursued. This robot design, called "UmbrellaBot" throughout the remainder of this document, consists of only three controllable joints. Two of the joints are prismatic beams that control a mechanism that extends and retracts a series of legs. The third joint is another prismatic beam connected between the other two joints in sequence, thus allowing the robot's body to expand and contract like an inchworm. A diagram presenting the preliminary design of the robot, referenced as

"UmbrellaBot" in this report, is shown in Figure 3. A more detailed illustration of one of the mechanism portions of such a robot is presented in Figure 4.

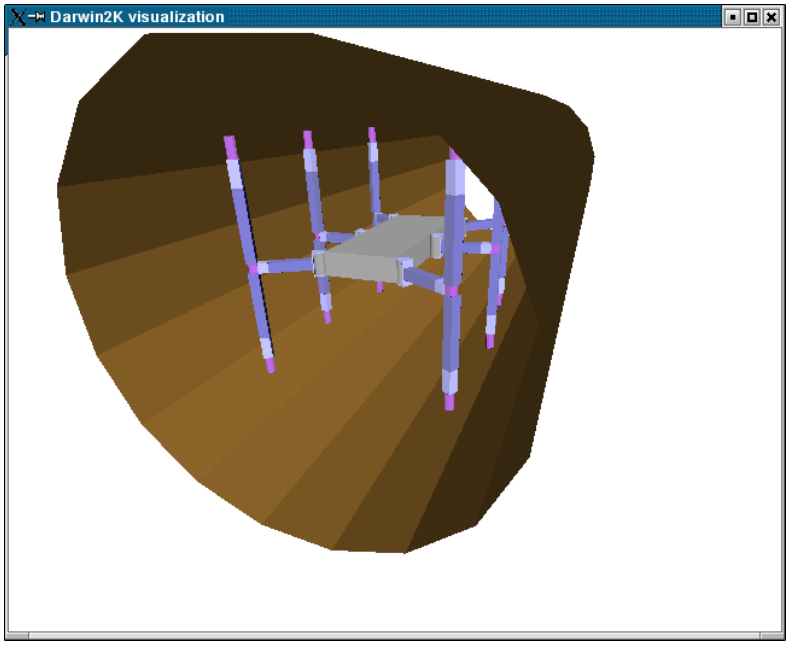

Figure 2 – Modified Darwin2K HexBot on Tube Terrain

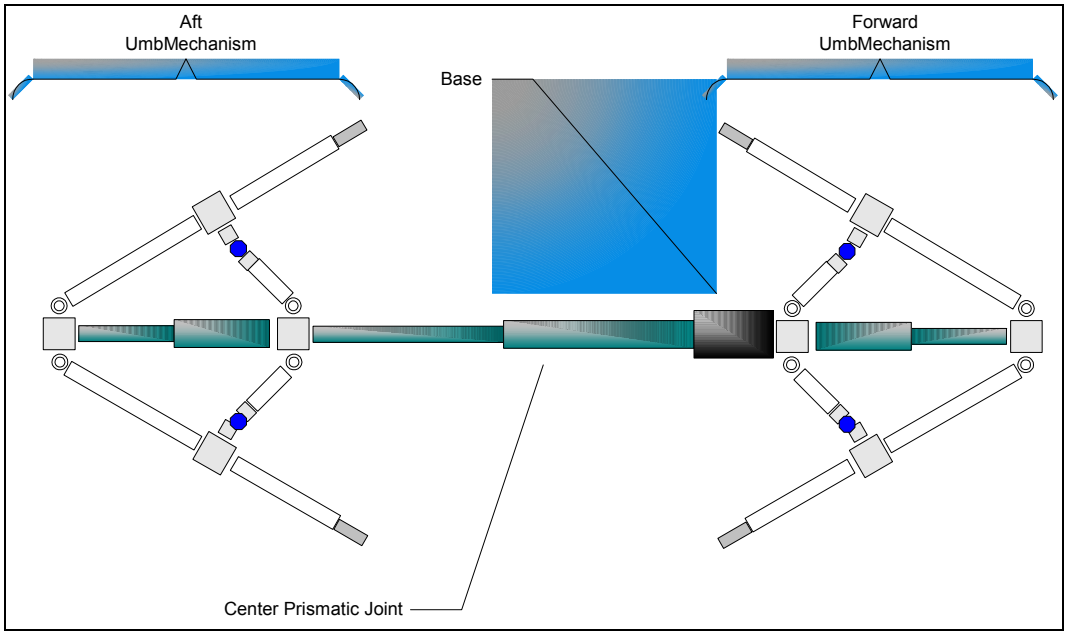

Figure 3 – UmbrellaBot Preliminary Design

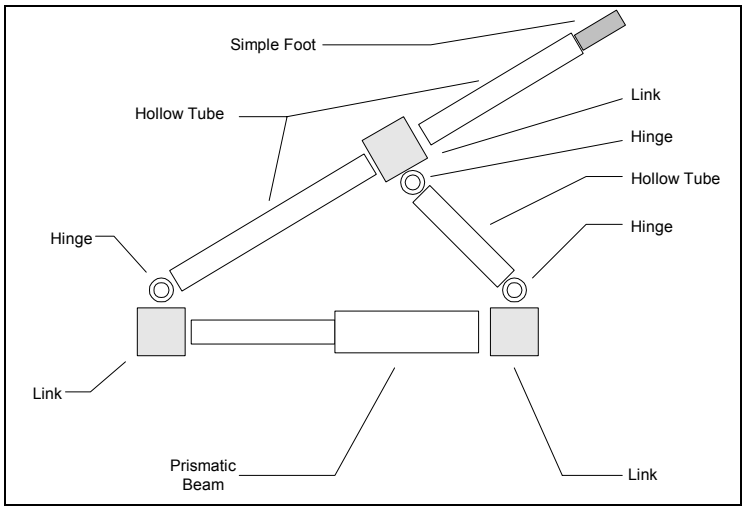

Figure 4 – Umbrella Leg Mechanism

Unfortunately, the leg design shown in Figure 4 results in a series of "closed chain" mechanisms, or mechanisms in which the dynamics calculations loop back upon themselves. In its unmodified form, Darwin2K cannot simulate the dynamics of such mechanisms. The following sections document the changes made to Darwin2K to implement the UmbrellaBot configuration.

5.0 Darwin2K Modifications

In Darwin2K, robots configurations are "assembled" using a map to connect the series of robotic joints (e.g., a prismatic beam), links (e.g., a hollow tube), bases, and end effectors to form a single design. Obviously, Darwin2K implements a finite number of types of components stored in a library. In order to build certain specific configurations, it is necessary to add required components, controllers, mechanisms, or other classes to the Darwin2K source code. It should be noted that the information presented in the following sections represents the work and experience of Dynamic Concepts, Inc., using Darwin2K versions 0.88, 0.90-Beta, 0.90, and 0.91a.

5.1 Links

The UmbrellaBot configuration required that a new link be incorporated into the Darwin2K database. This link, called a "cubeLink," has six interfaces for connection, rather than the standard two. The cubeLink class was added to the "links.h", "links.cxx", and "builtInSynTypes.cxx" files in the darwin2k-0.90.beta/src/d2k/modules directory. The source

code modifications are shown in Appendix A. It should be noted that there is an error in the autoconf/automake associated with Darwin2K versions 0.88 through 0.90. In order to recompile the source code, a special script file was developed to explicitly delete the executables, object files, libraries, and configuration "cache" files associated with the Darwin2K software.

5.2 Joints

In the Darwin2K context, joints are considered to be configuration connectors with at least one degree of freedom that is driven by a motor. In order to drive a robot, a control algorithm must be applied to the robot's joints. It was necessary to develop a new class of joint to produce the umbrella robot. This new joint class, called "hingeJoint," is not a true robotic joint. Rather, it is a simple hinge used to form the mechanism that extends the robotic legs of the UmbrellaBot. There is no motor or gear associated with the hingeJoint class. The hingeJoint class was added to the "componentJoint.h" and "componentJoint.cxx" files in the darwin2k-0.90.beta/src/d2k/modules directory and the "builtInSynTypes.cxx" file in the darwin2k-0.90.beta/src/d2k/db directory. Modifications to the files are shown in Appendix A.

5.3 Mechanisms and Evaluator

The most difficult aspect of creating the UmbrellaBot has been the incorporation of the series of "closed chain" mechanisms described previously, the means by which robot's legs are extended. These mechanisms are created by applying a series of pin constraints to the individual components of the configuration. Mechanism constraints are incorporated in Darwin2K by creating either a component or an evaluator. Components are very much like the Darwin2K links and joints discussed previously. Components may be created by adding class definitions and methods to existing code and declaring those new classes to the Darwin2K database. Components are instantiated by adding relevant sections to the parameter file (p-file) for the given configuration. Evaluators are treated somewhat differently. Evaluators are derived directly from the d2kSimulator class. The mechanism constraints have been added to the UmbrellaBot configuration by deriving a new evaluator class called "umbMechanismEvaluator." The source code defining the umbMechanismEvaluator class is presented in Appendix A.

In order to implement the umbMechanismEvaluator class, a "plug-in" has been added to the Darwin2K source code. A plug-in is a tool, component, or constraint stored in a special

library and linked to the Darwin2K source code. This task was accomplished by using the "darwin2k-makeplugin" command in version 0.90 beta. Note that, in order for this command to work properly, the directory with the appropriate Darwin2K executables must be in the user's path. The darwin2k-makeplugin command creates a series of classes and libraries that automatically link to the darwin2k executables. The plug-in method utilizes the Darwin2K evaluator approach discussed previously. The constraint plug-in applies a planar pin constraint to end points specified in the parameter file associated with the simulation. A diagram of the umbMechanismEvaluator constraints implemented in Darwin2K is presented in Figure 5.

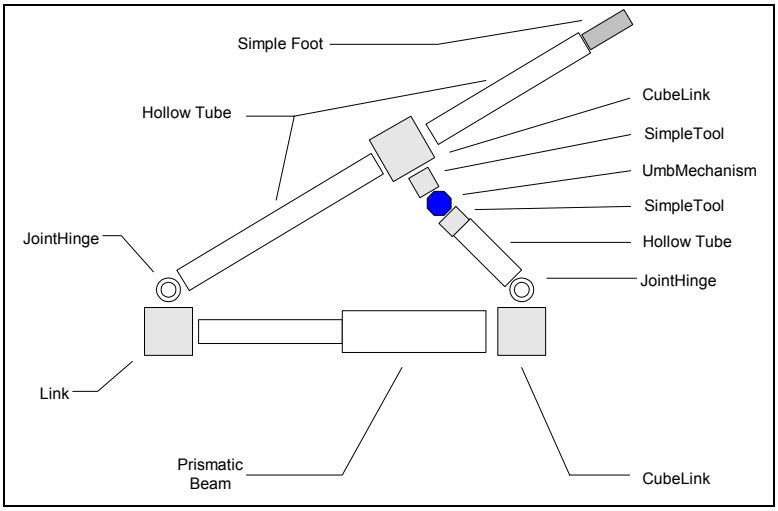

Figure 5 –Funtional Diagram of umbMechanism Class

It should be noted that, in order for the plug-in to compile and link properly with the rest of the Darwin2K source code, the "config.cache" file associated for a particular plug-in must be deleted manually. A diagram of the Darwin2K simulation of the mechanism is presented in Figure 6. With the successful creation of the "closed chain" mechanism, it is now possible to use Darwin2K simulate the dynamics of the UmbrellaBot and investigate the possibilities of evolving certain portions of the robot's configuration to improve its overall performance.

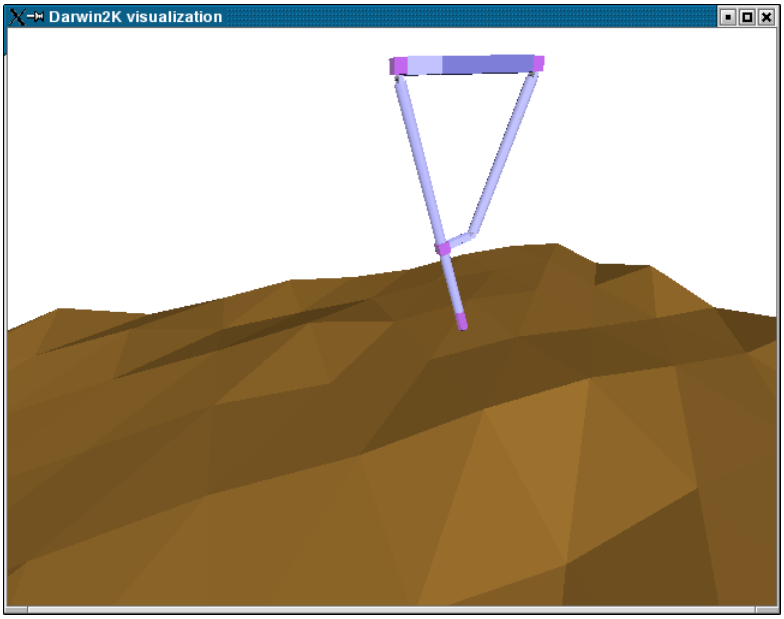

Figure 6 – Darwin2K Simulation implementing umbMechanism

5.4 Controller

An algorithm to control the prismatic joints of the UmbrellaBot has been developed. This controller class, called "umbController," has been derived from the Darwin2K "jointController" class and implements a proportional / integral / derivative (PID) control scheme. Gains for the controller are read from the evaluator parameter file. A complete listing of the umbController source code is presented in Appendix A.

6.0 Simulation and Evaluation Procedure

In order to evaluate a robot configuration using Darwin2K, a specific sequence of actions must be followed in order to ensure that the program will properly analyze the correct configuration. This sequence of analyses is discussed in the following sections, along with examples of the input required for each step and examples of graphics generated by the Darwin2K suite of tools.

6.1 Configuration Display (displayCfgGL)

When building a prototype in Darwin2K, it is often useful to check the robot configuration to ensure the joints and links are being constructed as intended. The program "displayCfgGL" is designed to accomplish this task. The program displayCfgGL graphically

renders the robot configuration and allows the user to view the specified configuration from virtually any angle, check the motion of the robotic joints, and inspect the dynamic chains formed by Darwin2K. A diagram presenting the X Windows screens generated by displayCfgGL are shown in Figure 7.

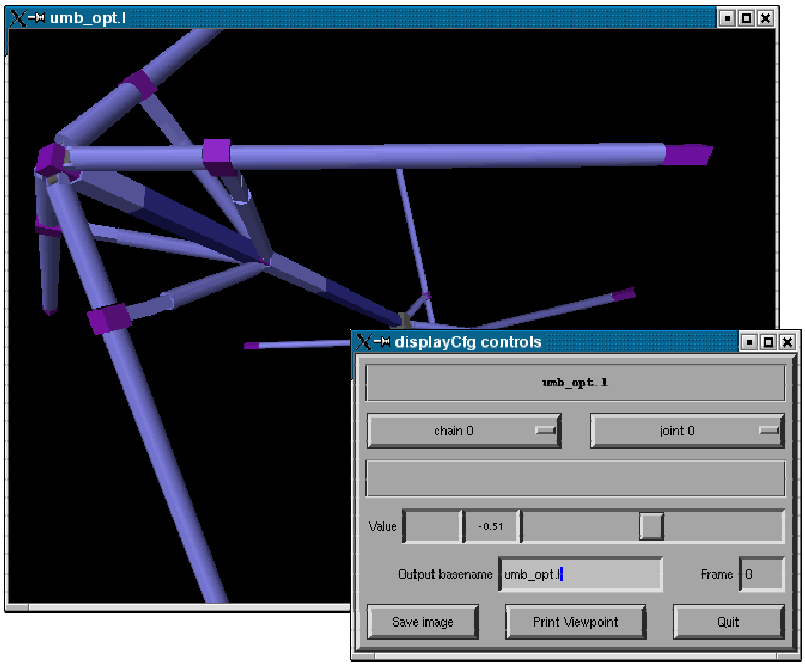

Figure 7 – Graphics Generated by Darwin2K displayCfgGL Program

The displayCfgGL program is activated by typing "`displayCfgGL config.l`" at the prompt of a Unix console window, where the file "config.l" is the configuration file of the desired robot.

6.2 Simulation (evStandaloneGL)

Before a robot configuration can be analyzed and refined by the Darwin2K population manager, it is necessary to simulate the dynamics of the robot's baseline configuration to ensure that the controller functions properly, that the robot does not collide with any objects, and that the robot can achieve it's goals. There are two dynamic simulation programs included in the most recent version of Darwin2K: the "d2kSimGL" simulation and the "evStandaloneGL" simulation. Both programs require a configuration file (config.l), an evaluation parameter file (eval.p), and a terrain definition file (terrain.dat). However, the evStandaloneGL program also requires a population manager parameter file (pm.p), which defines the robot's performance

metrics. The d2kSimGL program is activated by typing "`d2kSimGL eval.p config.l`" at the prompt of a Unix console window, while evStandaloneGL is activated by typing "`evStandaloneGL pm.p eval.p config.l`" at the prompt of a Unix console window. Both programs graphically render the simulated dynamics of the robot. A diagram presenting an example of the graphics generated by the Darwin2K dynamic simulations is shown in Figure 8.

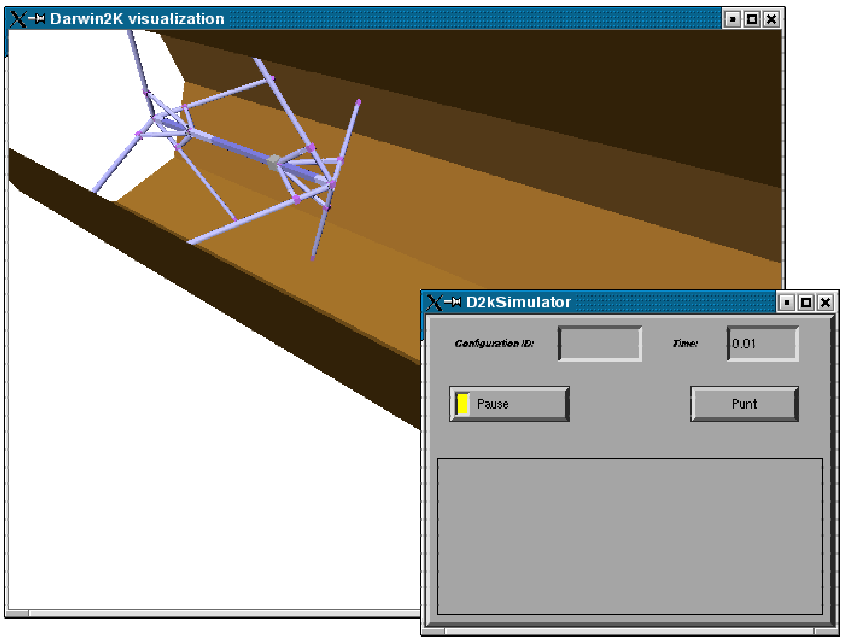

Figure 8 – Graphics Generated by Darwin2K d2kSimGL/evStandaloneGL Program

6.3 Population Manager (pmStandalone)

The Darwin2K population manager program, "pmStandalone," is used to generate many different robot configurations based on the baseline configuration, evaluate each configuration based on the performance metrics defined in the population manager parameter file (pm.p), and determine the optimal robot configuration for each performance metric. Unlike the d2kSimGL and evStandaloneGL programs, the pmStandalone program does not generate any graphic images of the robot dynamics. The pmStandalone program is activated by typing "`pmStandalone pm.p eval.p`" at the prompt of a Unix console window

7.0 Results

The Darwin2K population manager has been used to optimize a portion the configuration of the UmbrellaBot. Many analytical iterations were necessary in order to properly configure Darwin2K to perform the optimization. The input files required for the Darwin2K population manager, the robot configuration file (umb_evt_3.l), the evaluation parameter file (umbEvalPM.p), the performance metric parameter file (pm.p), and the terrain definition file (cyl3.dat), are presented in Appendix B. The task given to the robot was to move one unit of length down the long axis of the tube, which generally required one complete motion cycle of the robot. The robot configuration was measured in five categories of performance: 1.) completion of the task, 2.) the mass of robot, 3.) the time required to complete the task, 4.) the power consumed by the robot, and 5.) the deflection of the robot's links. The program manager was allowed to run through 10 generations with 10 population members each, for a total of 100 configurations. The population manager was allowed to vary (within limits) the length of the center prismatic joint and the length of the long "hollowTube" portion of the robot's legs. Three "optimal" robot configurations have been generated by the Darwin2K program manager. A listing of the population manager results is presented in Figure 9 and the optimal configuration with respect to the time metric is presented in Figure 10. The population manager calculated that, for the time performance metric, the best length of the center prismatic joint to be 1.1875, the length of the front leg hollowTube link to be 1.4125, and the length of the rear leg hollow Tube to be 1.405 (units are not provided since the robot's configuration was defined relative only to the tube). It is highly unlikely that this solution would have been reached using classical analysis methods. Provide with more resources, other parameters could be optimized, including the robot's controller gains or the number of legs for robot locomotion (or both).

```
Logged population.
100
  totalCfgs      10
  optimalCfgs    3
  feasibleCfgs   10
  evaluators     0
  weights: [0] 0.000
  100 [0] 1.000000 taskCompletionMetric            best = 1.000
  100 [1] 0.000233 massMetric                      best = 12.139
  100 [2] 0.002838 timeMetric                      best = 8.185
  100 [3] 0.242532 powerMetric                     best = 201.050
  100 [4] 0.001825 linkDeflectionMetric            best = 0.010
Generated 100 configurations, max = 100
Done.
```

Figure 9 – Output Log of the Darwin2K pmStandalone Program

```
((hexapodBase ((const 0.15 0.15 2 0)
(const 0.15 0.15 2 0)
(const 0.15 0.15 2 0)
(const 0.001 0.001 2 0))
 ((const 4 (1 0 left (const 0 270 2 2)))
(const 5 (2 0 right (const 0 270 2 2)))))
(prismaticBeam2M lowInertia ((const 0 1 3 2)
(const 0 1 5 10)
(const 0 1 2 2)
(const 0 1 2 0)
(const 0.02 0.15 3 3)
(const 0.001 0.005 3 0)
(var 1 1.5 3 3))
 ((const 1 (14 0 inherit (const 0 270 2 0)))))
(cubeLink hollowTube ((const 0 1 2 0)
(const 0.005 0.08 3 7)
(const 0.005 0.08 3 7)
(const 0.0015 0.005 3 7))
 ((const 1 (5 0 inherit (const 0 270 2 0)))
(const 2 (3 0 inherit (const 0 270 2 0)))
(const 3 (3 0 inherit (const 0 270 2 0)))
(const 4 (3 0 inherit (const 0 270 2 0)))
(const 5 (3 0 inherit (const 0 270 2 0)))))
(hingeJoint freeRevoluteJoint ((const 0 1 2 2)
(const 0 1 2 10)
(const 0 1 2 0)
(const -1.57 1.57 6 52)
(const 0.005 0.015 3 3)
(const 0 0.2 2 0)
(const 0.01 0.01 2 0)
(const 0.05 0.05 2 0))
 ((const 1 (4 0 inherit (const 0 270 2 0)))))
(hollowTube hollowTube ((const 0 1 2 0)
(const 0.55 0.55 2 0)
(const 0.02 0.15 3 3)
(const 0.001 0.005 3 0))
 ((const 1 (11 0 inherit (const 0 270 2 0)))))
(prismaticBeam2M lowInertia ((const 0 1 3 2)
(const 0 1 5 10)
(const 0 1 2 2)
(const 0 1 2 0)
(const 0.02 0.15 3 3)
(const 0.001 0.005 3 0)
(const 0.05 0.7 3 6))
 ((const 1 (6 1 inherit (const 0 270 2 0)))))
(cubeLink hollowTube ((const 0 1 2 0)
(const 0.005 0.08 3 7)
(const 0.005 0.08 3 7)
(const 0.0015 0.005 3 7))
 ((const 2 (7 0 inherit (const 0 270 2 0)))
(const 3 (7 0 inherit (const 0 270 2 0)))
(const 4 (7 0 inherit (const 0 270 2 0)))
(const 5 (7 0 inherit (const 0 270 2 0)))))
(hingeJoint freeRevoluteJoint ((const 0 1 2 2)
(const 0 1 2 10)
(const 0 1 2 0)
(const -1.57 1.57 6 41)
(const 0.005 0.015 3 3)
(const 0 0.2 2 0)
(const 0.01 0.01 2 0)
(const 0.05 0.05 2 0))
 ((const 1 (8 0 inherit (const 0 270 2 0)))))
(hollowTube hollowTube ((const 0 1 2 0)
(const 0.39 0.39 2 0)
(const 0.02 0.15 3 3)
(const 0.001 0.005 3 0))
 ((const 1 (9 0 inherit (const 0 270 2 0)))))
(cubeLink hollowTube ((const 0 1 2 0)
(const 0.005 0.08 3 7)
(const 0.005 0.08 3 7)
(const 0.0015 0.005 3 7))
 ((const 1 (12 0 inherit (const 0 270 2 0)))
(const 4 (11 0 inherit (const 0 270 2 0)))))
(hingeJoint freeRevoluteJoint ((const 0 1 2 2)
(const 33 33 6 63)
(const 0 1 2 0)
(const -1.57 1.57 6 20)
(const 0.005 0.015 3 3)
(const 0 0.2 2 0)
(const 0.01 0.01 2 0)
(const 0.05 0.05 2 0))
 ((const 1 (11 0 inherit (const 0 270 2 0)))))
(simpleTool ((const 0.05 0.1 3 7))
 ())
(hollowTube hollowTube ((const 0 1 2 0)
(var 1.405 1.415 2 0)
(const 0.02 0.15 3 3)
(const 0.001 0.005 3 0))
 ((const 1 (13 0 inherit (const 0 270 2 0)))))
(rubberFoot ((const 0.01 0.6 4 4)
(const 0.005 0.08 4 15)
(const 0.0015 0.005 3 7)
(const 500 4000 4 12)
(const 10 500 4 0))
 ())
(cubeLink hollowTube ((const 0 1 2 0)
(const 0.005 0.08 3 7)
(const 0.005 0.08 3 7)
(const 0.0015 0.005 3 7))
 ((const 1 (17 0 inherit (const 0 270 2 0)))
(const 2 (15 0 inherit (const 0 270 2 0)))
(const 3 (15 0 inherit (const 0 270 2 0)))
(const 4 (15 0 inherit (const 0 270 2 0)))
(const 5 (15 0 inherit (const 0 270 2 0)))))
(hingeJoint freeRevoluteJoint ((const 0 1 2 2)
(const 0 1 2 10)
(const 0 1 2 0)
(const -1.57 1.57 6 11)
(const 0.005 0.015 3 3)
(const 0 0.2 2 0)
(const 0.01 0.01 2 0)
(const 0.05 0.05 2 0))
 ((const 1 (16 0 inherit (const 0 270 2 0)))))
(hollowTube hollowTube ((const 0 1 2 0)
(const 0.55 0.55 2 0)
(const 0.02 0.15 3 3)
(const 0.001 0.005 3 0))

 ((const 1 (23 0 inherit (const 0 270 2 0)))))
(prismaticBeam2M lowInertia ((const 0 1 3 2)
(const 0 1 5 10)
(const 0 1 2 2)
(const 0 1 2 0)
(const 0.02 0.15 3 3)
(const 0.001 0.005 3 0)
(const 0.05 0.7 3 6))
 ((const 1 (18 0 inherit (const 0 270 2 0)))))
(cubeLink hollowTube ((const 0 1 2 0)
(const 0.005 0.08 3 7)
(const 0.005 0.08 3 7)
(const 0.0015 0.005 3 7))
 ((const 2 (19 0 inherit (const 0 270 2 0)))
(const 3 (19 0 inherit (const 0 270 2 0)))
(const 4 (19 0 inherit (const 0 270 2 0)))
(const 5 (19 0 inherit (const 0 270 2 0)))))
(hingeJoint freeRevoluteJoint ((const 0 1 2 2)
(const 0 1 2 10)
(const 0 1 2 0)
(const -1.57 1.57 6 41)
(const 0.005 0.015 3 3)
(const 0 0.2 2 0)
(const 0.01 0.01 2 0)
(const 0.05 0.05 2 0))
 ((const 1 (20 0 inherit (const 0 270 2 0)))))
(hollowTube hollowTube ((const 0 1 2 0)
(const 0.39 0.39 2 0)
(const 0.02 0.15 3 3)
(const 0.001 0.005 3 0))
 ((const 1 (21 0 inherit (const 0 270 2 0)))))
(cubeLink hollowTube ((const 0 1 2 0)
(const 0.005 0.08 3 7)
(const 0.005 0.08 3 7)
(const 0.0015 0.005 3 7))
 ((const 1 (24 0 inherit (const 0 270 2 0)))
(const 5 (23 0 inherit (const 0 270 2 0)))))
(hingeJoint freeRevoluteJoint ((const 0 1 2 2)
(const 33 33 6 63)
(const 0 1 2 0)
(const -1.57 1.57 6 13)
(const 0.005 0.015 3 3)
(const 0 0.2 2 0)
(const 0.01 0.01 2 0)
(const 0.05 0.05 2 0))
 ((const 1 (23 0 inherit (const 0 270 2 0)))))
(simpleTool ((const 0.05 0.1 3 7))
 ())
(hollowTube hollowTube ((const 0 1 2 0)
(var 1.405 1.415 2 2)
(const 0.02 0.15 3 3)
(const 0.001 0.005 3 0))
 ((const 1 (25 0 inherit (const 0 270 2 0)))))
(rubberFoot ((const 0.01 0.6 4 4)
(const 0.005 0.08 4 15)
(const 0.0015 0.005 3 7)
(const 500 4000 4 12)
(const 10 500 4 0))
 )))
```

Figure 10 – Time Metric Optimal UmbrellaBot Configuration File

8.0 Summary

Efforts associated with the Space RATS project are documented in this report. The design of a tube crawling robot is presented. The design of the robot has been analyzed using Darwin2K, a suite of tools designed for robot simulation and optimization. A brief overview of Darwin2K is discussed, and the modifications made to Darwin2K in order to perform the analyses are documented. The evolutionary analysis tools provided by Darwin2K have been used to optimize portions of the robot's configuration for certain performance metrics.

The Darwin2K software toolkit has great potential for automating robotic design synthesis and optimization. However, the relative difficulty in using Darwin2K in its present form probably precludes its widespread adoption until several "ease of use" issures are addressed. However, the primary author of the software is presently working to address these issues.

The tube crawler robot whose design was simulated and ultimately refined using Darwin2K has been developed to the point that a more complete analysis is now feasible. Additional design studies will be required to determine appropriate sensors, power supplies, and useability constraints.

APPENDIX A

Darwin2K Source Code Modifications

Links.H

```
class cubeLink : public linkModule
{
  public:
    MAKE_COVER_FUNCTIONS(cubeLink);
    DECLARE_PARENT_CLASS(linkModule);
    material mat;
    double I, J, A;    // moments used for computing deflections
    virtual int createGeometry(handedness which);
    virtual int numParams(void) const { return 4; }
    virtual paramSpec * getParamSpec(void) const;
    virtual int numConnectors(void) const { return 6; }
    virtual connectorSpec * getConnectorSpec(void) const;
    virtual int requiresContext(void) const { return 1; }
    virtual int computeDeflections(ownerSegment *ms, linkForceContext *lf,
                triple &trans, quaternion &rot);
};
```

Links.CXX

```
DEFINE_CLASS_ID(cubeLink);
//
//  cubelink methods
//

paramSpec * cubeLink::getParamSpec(void) const {
  static paramSpec specs[] = {
    { "sel-material",  "material selection (int)",
      0.0, NULL, NULL, "material" },
    { "length",        "length (m)",         0.0, NULL, NULL, NULL },
    { "diameter",      "outer diameter (m)", 0.4, NULL, NULL, NULL },
    { "thickness",     "wall thickness (m)", 0.1, NULL, NULL, NULL },
    { NULL, NULL, 0.0, NULL, NULL, NULL }
  };
  return specs;
}

connectorSpec * cubeLink::getConnectorSpec(void) const {
  static connectorSpec specs[] = {
    { "connector-1", "connector to base of link (??)" },
    { "connector-2", "connector to end of link (??)" },
    { "connector-3", "connector to end of link (??)" },
    { "connector-4", "connector to end of link (??)" },
    { "connector-5", "connector to end of link (??)" },
    { "connector-6", "connector to end of link (??)" },
    { NULL, NULL }
  };
  return specs;
}

int cubeLink::createGeometry(handedness which) {
  const component *temp;
  int errors = 0;

  if (!context) {
    logError("cubeLink::createGeometry");
    logPrintf("  component context required.\n");
    return 0;
  }

  GET_COMPONENT(mat, material, 0);

  if (errors) return 0;

  double l = getParam(1)->val;
  double orad = getParam(2)->val/2;
  double t = getParam(3)->val;
  double d = orad*2;
```

```
  double ir = orad-t;

  if (l < d) l = d;

  as = new assembly;

  //
  // make 1st part
  //
  part *p1 = new part;

  as->addPart(p1);
  as->root = p1;
  p1->name = "cubeLink";
  mpoly *bBody = createBlock(triple(d, d, d),
                             triple(-d/2,-d/2, 0),
                             mat.density, G_PURPLE1);
  p1->addMPoly(bBody);

  connector *c1 = new connector(p1);
  c1->frame = tmatrix(M_PI/2, 0, 0, 0, -orad-t, orad);
  as->addConnector(c1);                // connectorID = 0

  connector *c2 = new connector(p1);
  c2->frame = tmatrix(-M_PI/2, M_PI/2, 0, 0, orad, orad);
  as->addConnector(c2);                // connectorID = 0

  connector *c3 = new connector(p1);
  c3->frame = tmatrix(0, 0, M_PI/2, 0, 0, l);
  as->addConnector(c3);                // connectorID = 1

  connector *c4 = new connector(p1);
  c4->frame = tmatrix(M_PI, 0, M_PI/2, 0, 0, 0);
  as->addConnector(c4);                // connectorID = 2

  connector *c5 = new connector(p1);
  c5->frame = tmatrix(M_PI/2, 0, M_PI/2, d/2, 0, l/2);
  as->addConnector(c5);                // connectorID = 2

  connector *c6 = new connector(p1);
  c6->frame = tmatrix(-M_PI/2, 0, M_PI/2, -d/2, 0, l/2);
  as->addConnector(c6);                // connectorID = 2

  // these assume a circular cross-section, not polygonal
  I = M_PI*0.25*(orad*orad*orad*orad - ir*ir*ir*ir);
  J = I*2;
  A = M_PI*(orad*orad - ir*ir);

  assignConnectorIDs();

  return 1;
}

int cubeLink::computeDeflections(ownerSegment *ms,
                                 linkForceContext *lf,
                                 triple &trans, quaternion &rot) {
  connector *c1 = ms->getConnector(0);
  connector *c2 = ms->getConnector(1);

  triple pt1 = c1->computePosition();
  triple pt2 = c2->computePosition();

  double d1 = lf->computeD(pt1);
  double d2 = lf->computeD(pt2);

  lf->computeDeflection(d1, d2, I, I, J, A, mat.e, mat.g,
                        trans, rot);

  return 1;
}
```

BuiltInSynTypes.CXX

```
ADD_DB_TYPE(module, cubeLink, NULL);
```

ComponentJoint.H:

```
class hingeJoint : public componentJoint
{
  public:
    MAKE_COVER_FUNCTIONS(hingeJoint);
    DECLARE_PARENT_CLASS(componentJoint);
    DECLARE_COMPONENT_MEMBERS(1);
    virtual int createGeometry(handedness which);
    virtual int numParams(void) const { return 8; }
    virtual paramSpec * getParamSpec(void) const;
    virtual int numConnectors(void) const { return 2; }
    virtual connectorSpec * getConnectorSpec(void) const;
    virtual int computeDeflections(ownerSegment *ms, linkForceContext *lf,
                triple &trans, quaternion &rot);
    virtual int postInstantiationInit(void);
};
```

ComponentJoint.CXX:

```
////////////////////////////////////////////////////////////////////////
//
// hingeJoint
//
////////////////////////////////////////////////////////////////////////

DEFINE_CLASS_ID(hingeJoint);
ptrList hingeJoint::depTab;

paramSpec * hingeJoint::getParamSpec(void) const {
  static paramSpec specs[] = {
    { "sel-motor",     "motor selection (int)",
      0.0, NULL, NULL, "rotaryActuator" },
    { "sel-gearbox",   "gearbox selection (int)",
      0.0, NULL, NULL, "gearBox" },
    { "sel-material",  "material selection (int)",
      0.0, NULL, NULL, "material" },
    { "joint-offset",  "joint angle offset (deg)", 0.0, NULL, NULL, NULL },
    { "wall-thick",    "wall thickness (m)",       0.01, NULL, NULL, NULL },
    { "offset-dist",   "offset distance (m)",      0.5, NULL, NULL, NULL },
    { "diameter",      "outer diameter (m)",       0.1, NULL, NULL, NULL },
    { "length",        "length (m)",               0.5, NULL, NULL, NULL },
    { NULL, NULL, 0.0, NULL, NULL, NULL }
  };
  return specs;
}

connectorSpec * hingeJoint::getConnectorSpec(void) const {
  static connectorSpec specs[] = {
    { "connector-1", "connector to base of first joint link (??)" },
    { "connector-2", "connector to end of second joint link (??)" },
    { NULL, NULL }
  };
  return specs;
}

int hingeJoint::createGeometry(handedness whichSide) {
  const component *temp;
```

```
  int errors = 0;
  //double angleOffset = getParam(3)->val;
  double t = getParam(4)->val;

  if (!context) {
    if (verboseLevel >= SO_ERRORS) {
      logError("offsetElbow::createGeometry");
      logPrintf("  component context required.\n");
    }
    return 0;
  }

  GET_COMPONENT(act, rotaryActuator, 0);
  GET_COMPONENT(gbx, gearBox, 1);
  GET_COMPONENT(mat, material, 2);

  if (errors) return 0;

  lim.computeLimits(&act, &gbx);

//  double r = act.diameter > gbx.diameter ? act.diameter/2 : gbx.diameter/2;
//  double l = act.length + gbx.length;
  double r = getParam(6)->val/2;
  double l = getParam(7)->val;

  double orad = computeOuterRadius(8, r, t);
  double d = getParam(5)->val + (t + orad)*1.001;

  mpoly *motorPoly = createMotorPoly(&act);
  mpoly *gearboxPoly = createGearheadPoly(&gbx);

  ptrList *tubeList = makeRegularHollowPrism(8, l, orad, t,
                                             mat.density, G_GREY1, 1);
  mpoly *cap1 = makeRoundedRectangle(orad, d, t, mat.density, G_BLUE2);
  mpoly *cap2 = makeRoundedRectangle(orad, d, t, mat.density, G_BLUE2);
  mpoly *bracket = createBlock(triple(2*orad, t, l),
                               triple(-orad, -d, 0), mat.density, G_BLUE2);

  mpoly *lowDetailTube = makeRegularPrism(lowDetailTubeNumSides,
                                          l-geomCollGap*2, orad,
                                          mat.density, G_GREY1, 1);
  transformPolyCoords(lowDetailTube, triple(0, 0, geomCollGap), quaternion(0));

  transformPolyCoords(cap1, triple(0, 0, l), quaternion(0));
  transformPolyCoords(cap2, triple(0, 0, -t), quaternion(0));
  transformPolyCoords(motorPoly, triple(0, 0, gbx.length), quaternion(0));

  part *p1 = new part;
  part *p2 = new part;

  p1->name = "offsetElbowTube";
  p2->name = "offsetElbowBracket";

  as = new assembly;
  as->addPart(p1);
  as->addPart(p2);
  as->root = p1;

  p1->addMPolyList(tubeList);
  delete tubeList;

  p1->addBPoly(lowDetailTube);
//  p1->addMPoly(gearboxPoly);
//  p1->addMPoly(motorPoly);

  p2->addMPoly(cap1);
  p2->addMPoly(cap2);
  p2->addMPoly(bracket);

  //if (whichSide == module::RIGHT) angleOffset = -angleOffset;
  connector *c1 = new connector(p1, 0);
```

```
  c1->frame = tmatrix(0, 0, 0, 0, 0, 0);
  // was c1->frame = tmatrix(0, 0, (angleOffset)*M_PI/180.0, 0, 0, 0);
  // was angleOffset+180

  connector *c2 = new connector(p2, 0);
  c2->frame = tmatrix(0, 0, 0, 0, 0, 0);
  if (whichSide == module::RIGHT) {
    c1->rotate(tmatrix::y, M_PI);
    c2->rotate(tmatrix::y, M_PI);
  }
  joint *jt =
    new joint(c1, c2, 0, joint::REVOLUTE, errors, this);
  jt->limits[0] = 1;
  jt->max(0) = M_PI/2;
  jt->min(0) = -M_PI/2;
  jt->dryFriction(0) = lim.dryFriction;
  jt->viscousFriction(0) = lim.damping;
  jt->inertia(0) = lim.actI;
  // was jt->max(0) = M_PI/2 - angleOffset*M_PI/180;
  // was jt->min(0) = -M_PI/2 - angleOffset*M_PI/180;
  /*
  if (whichSide == module::RIGHT) {
    double mtemp = -jt->min(0);
    jt->min(0) = -jt->max(0);
    jt->max(0) = mtemp;
  }
  */
  c1 = new connector(p1);
  c1->frame = tmatrix(-M_PI/2, 0, 0, 0, orad, l/2);
  as->addConnector(c1);
  c2 = new connector(p2);
  c2->frame = tmatrix(M_PI/2, 0, 0, 0, -d, l/2);
  as->addConnector(c2);
  assignConnectorIDs();
  return (errors == 0);
}

int hingeJoint::postInstantiationInit(void) {
  joint *jt = getJoint(0);
  jt->pos(0) = getParam(3)->val;
  if (jt->pos(0) < jt->min(0)) jt->pos(0) = jt->min(0);
  else if (jt->pos(0) > jt->max(0)) jt->pos(0) = jt->max(0);
  return 1;
}

int hingeJoint::computeDeflections(ownerSegment *ms,
                                   linkForceContext *lf,
                                   triple &trans, quaternion &rot) {
  trans = 0;
  rot = 0;
  return 1;
}

int hingeJoint::initializeDependencyTable(void) {
  if (depTab.n > 0) return 0;

  ptrList *l = new ptrList(1);
  l->addPtr(0);          // gearbox depends on motor (param 0)

  depTab.addPtr(NULL);   // motor
  depTab.addPtr(l);      // gearbox
  depTab.addPtr(NULL);   // material
  depTab.addPtr(NULL);   // o.d.
  depTab.addPtr(NULL);   // wall thickness
  depTab.addPtr(NULL);   // offset distance
  depTab.addPtr(NULL);   // outer diameter
  depTab.addPtr(NULL);   // length

  return 1;
}
```

umbMechanismEval.H:

```
//########################################################################
//
// UMBMECHANISM - An evaluation module for Darwin2K
// Copyright (C) 2002 mcraft mcraft@dynamic-concepts.com
//
// PUT LICENSE HERE
//########################################################################

#ifndef INCumbMechanismEval_h
#define INCumbMechanismEval_h
#include <darwin2k/paramParser.h>

class umbMechanismController;

class umbMechanismEvaluator : public evaluator {
public:
  umbMechanismEvaluator(void);
  ~umbMechanismEvaluator(void);

  // this creates methods and members needed for runtime typing
  MAKE_COVER_FUNCTIONS(umbMechanismEvaluator);

  // this is also needed for runtime typing.  put in the proper
  // parent class if this class is not directly derived from evaluator.
  DECLARE_PARENT_CLASS(evaluator);

  //
  // data members
  //
  int maxConstraints;
  int firstPt[100];
  int secondPt[100];
  int numConstraints;
  int firstPoint, secondPoint;

  double goalX;
  double goalY;
  triple goal;
  triple goalVector;

  //
  // the d2kSimulator to which the d2kComponent belongs.  This is not set
  // until simInit() is called.
  //
  d2kSimulator *sim;

  //########################################
  // methods inheritied from evaluator()
  //########################################
  virtual int readParams(const char *fname);
//  virtual int readParams( paramParser *parser );
  virtual int postComponentCreationHook(void);
  virtual int init(ptrList *Cfgs);
  virtual int cleanup(void);
  virtual int evaluateConfiguration(void);
};

#endif /* INCumbMechanismEval_h */
```

umbMechanismEval.CXX:

```
////////////////////////////////////////////////////////////////////////
//
// UMBMECHANISM - An evaluation plugin for Darwin2K
// Copyright (C) 2002 mcraft mcraft@dynamic-concepts.com
//
// PUT LICENSE HERE
////////////////////////////////////////////////////////////////////////

#include "umbMechanismLocal.h"
#include "umbMechanismEval.h"
#include "umbMechanismModules.h"
#include "umbController.h"

DEFINE_CLASS_ID(umbMechanismEvaluator);

// constructor
umbMechanismEvaluator::umbMechanismEvaluator(void) {
}

// destructor
umbMechanismEvaluator::~umbMechanismEvaluator(void) {
}

// this method reads parameters from the specified file.
int umbMechanismEvaluator::readParams(const char *filename) {
//int umbMechanismEvaluator::readParams(paramParser *parser) {
  // if this class is not directly derived from evaluator, replace
  // evaluator:: with <parentClass>::
  int retval = evaluator::readParams(filename);
//  int foo;
//  double bar;
  int optional;

//  retval &= evaluator::readParams(filename);

  if (!theSimParser.pushContext("umbMechanismEvaluator")) {
    return 0;
  }

  // read the required parameters
  maxConstraints = 100;
  theSimParser.GET_INT(numConstraints);
  logPrintf("umbMechanismEval: numConstraints =  %d\n", numConstraints);
  if ( (numConstraints < 0) || (numConstraints > maxConstraints) )
  {
    logError("umbMechanismEvaluator::readParams");
    logPrintf("  numConstraints out of range! (0<numConstraints<100)\n");
    return 0;
  }

//
//  added for performance metrics
//
  theSimParser.GET_DOUBLE(goalX);
  theSimParser.GET_DOUBLE(goalY);
  goal = triple(goalX, goalY, 2.0);
  logPrintf("umbMechanismEval: goal =  ");
  PRINT_TRIPLE( goal );

//  for (int i=0; i<numConstraints; i++)
//  {
//    firstPt[i] = 1;
//    secondPt[i] = 2;
//    theSimParser.GET_INT(firstPt[i]);
//    theSimParser.GET_INT(secondPt[i]);
//    theSimParser.GET_INT(firstPoint);
//    theSimParser.GET_INT(seconint firstPt0 = 1;
//    firstPt[i]  = firstPoint;
//    secondPt[i] = secondPoint;
```

```
//    logPrintf("umbMechanismEval: firstPt%d  =  %d\n", i, firstPt[i]);
//    logPrintf("umbMechanismEval: secondPt%d =  %d\n", i, secondPt[i]);
//  }
//  firstPt[0] = 1;
//  secondPt[0] = 2;
  FILE *fp;

  const char *pointFile = "/home/mcraft/spacerats/v0.90/points.dat";

  if (theSimParser.GET_STRING(pointFile)) {
    pointFile = copyString(pointFile);
  }

  fp = fopen(pointFile,"r");

  if (!fp) {
    logError("umbMechanismEvaluator::readParams");
    logPrintf("  couldn't open point file %s\n", pointFile);
    return 0;
  }
  for (int i=0; i<numConstraints; i++)
  {
    fscanf(fp,"%d %d", &firstPt[i], &secondPt[i]);
    logPrintf("umbMechanismEval: firstPt%d  =  %d\n", i, firstPt[i]);
    logPrintf("umbMechanismEval: secondPt%d =  %d\n", i, secondPt[i]);
  }
  fclose(fp);

//    theSimParser.GET_INT(firstPoint);
//    theSimParser.GET_INT(secondPoint);
//    logPrintf("umbMechanismEval: firstPoint  =  %d\n", firstPoint);

//  retval &= theSimParser.GET_DOUBLE(bar);

  // read the optional parameters.  The QGET_* macros won't complain
  // if their parameter isn't defined in the p-file
  theSimParser.QGET_INT(optional);

  if (!retval) {
    logError("umbMechanismEvaluator::readParams");
    logPrintf("  failed to parse parameter file %s\n", filename);
  }

/*  double kp;
  double kv;
  double ki;

  if (!d2kComponent::readParams(parser)) {
    return 0;
  }

  parser->QGET_DOUBLE(kp);
  parser->QGET_DOUBLE(kv);
  parser->QGET_DOUBLE(ki);

  return 1; */

  return retval;
}

int umbMechanismEvaluator::postComponentCreationHook(void) {
  // if this class is not directly derived from d2kSimulator, replace
  // evaluator:: with <parentClass>::
  int retval = d2kSimulator::postComponentCreationHook();

  // do class-specific stuff here.  set retval to 0 if something fails.

  // if we want to find a payload in the list of d2kComponents,
  // we'd do this:
  payload *p = getComponentByType(payload, 1);
```

```
  if (!retval) {
    logError("umbMechanismEvaluator::postComponentInit");
    logPrintf("  something went wrong! (see above for possible messages\n");
  }

  return retval;
}


int umbMechanismEvaluator::init(ptrList *Cfgs) {
  // if this class is not directly derived from evaluator, replace
  // evaluator:: with <parentClass>::
  int retval = evaluator::init(Cfgs);

  // allocate stuff and perform configuration-dependent initialization.
  // note that evaluator::init() will set cfg to Cfg.

  for (int i=0; i<numConstraints; i++)
  {
    //
    //  endPointRecs keep track of tools, TCPs, etc., for a configuration
    //
    endPointRec *endPt1 = cfg->getEndPoint( firstPt[i] );
    endPointRec *endPt2 = cfg->getEndPoint( secondPt[i] );

    link *link1 = endPt1->l;
    link *link2 = endPt2->l;

    endPt1->tool->computeLinkCoords();
    triple bodyPt1 = endPt1->tool->xlink;

    endPt2->tool->computeLinkCoords();
    triple bodyPt2 = endPt2->tool->xlink;

    triple pinDirection_w;  // pin direction in world coords
    triple pinDirection_l;  // pin direction in link coords
    joint *jt;

    //
    //  get the link corresponding to endpt1 and see if there's a joint
    //  above it
    //
    if (link1->parentJoint)
      jt= link1->parentJoint;
    else if (link2->parentJoint)
      jt= link2->parentJoint;
    else
    {
      logError("umbMechanismEvaluator::init");
      logPrintf("  unable to locate a parentJoint for either end of the hinge mechanism!\n");
      return 0;
    }
    pinDirection_w = jt->axis[0];

    //
    //  pin direction is in world coords, so transform it in
    //  link1's coords with a quaternion
    //  link::Q transforms orientations from link to world, so use complement
    //
    quaternion Q_wtol = ~(link1->Q);
    pinDirection_l = Q_wtol*pinDirection_w*~Q_wtol;

    //
    //  build planarPinConstraint
    //
    planarPinConstraint *ppc
         = new planarPinConstraint( link1, bodyPt1, link2, bodyPt2, pinDirection_l );

    //
    //  add constraint to sim
    //
```

```
    if ( !ds->addObject( ppc ) )
    {
      logError("umbMechanismEvaluator::init");
      logPrintf("  unable to add constraint to simulation!\n");
      return 0;
    }
    logPrintf("umbMechanism:added constraint to sim!\n");
  }

  logPrintf("umbMechanismEval: numDOFs      =  %d\n", cfg->numDOFs );
  logPrintf("umbMechanismEval: numEndPoints =  %d\n", cfg->endPoints->n );

  triple coord;
  for (int i=0; i<cfg->endPoints->n; i++)
  {
    coord = cfg->getEndPoint(i)->xworld;
    logPrintf("umbMechanismEval: cfg->endpoint[%d]: %lf %lf %lf \n", i,
      coord.v[0], coord.v[1], coord.v[2]);
//    cfg->getEndPoint(i)->xworld->v[0],
//    cfg->getEndPoint(i)->xworld->v[1],
//    cfg->getEndPoint(i)->xworld->v[2]);
  }

/*    part *part1;
    part *part2;

    if ( !(link1->parts->n > 0 ) || !(link2->parts->n > 0 ) )
    {
      logError("umbMechanismEvaluator::init");
      logPrintf("  unable to locate parts!\n");
      return 0;
    }

    part1 = (part *) link1->parts->el(0);
    part2 = (part *) link2->parts->el(0);

  }

  logPrintf("umbMechanism:numConstraints = %d \n", numConstraints);
*/
  return retval;
}

int umbMechanismEvaluator::cleanup(void) {
  // de-allocate anything allocated in init()

  // if this class is not directly derived from evaluator, replace
  // evaluator:: with <parentClass>::
  return evaluator::cleanup();
}

// evaluates the configuration passed to init()
int umbMechanismEvaluator::evaluateConfiguration(void) {
  int retval = 1;
  int done = 0;
  bool firstPass = true;
  double startPos;
  double pos;
  double percentDone;

  taskCompletionMetric *tcm;
  timeMetric *tm;
  linkDeflectionMetric *ldm;

  // reset the metrics, as this may not be the first evaluation
  resetMetrics(&metrics);
  tcm = getMetricByName( taskCompletionMetric );
  tm  = getMetricByName( timeMetric );
  ldm = getMetricByName( linkDeflectionMetric );

  // apply the non-state-dependent metrics
```

```
  applyMetrics(&metrics, this, 1, 0);

  if (ldm)
    cfg->computeLockedTorques = 1;
  else
    cfg->computeLockedTorques = 0;

  // we might want to loop until some condition is met (i.e. done == 1),
  // or until there's a timeout (either in realtime or in simulation)

  while (!done && !checkTimeout() && ds->currentTime() < simTimeout)
  {
    // take a simulation step
    ds->stepForward(dt);

    // update the display and see if the user terminated the simulation
    if (updateDisplay()) break;

    // apply the state-dependent metrics
    applyMetrics(&metrics, this, 0, 1);

    // we may also want to set 'done' if we've detected some termination
    // condition (either success or failure)
    triple o = cfg->mech->root->T(triple(0));
    triple xb(cfg->mech->root->r(0, 0), cfg->mech->root->r(1, 0), cfg->mech->root->r(2, 0));
    triple yb(cfg->mech->root->r(0, 1), cfg->mech->root->r(1, 1), cfg->mech->root->r(2, 1));
    triple zb(cfg->mech->root->r(0, 2), cfg->mech->root->r(1, 2), cfg->mech->root->r(2, 2));

    goalVector = goal-o;

    if (firstPass)
    {
      startPos = abs(goalVector);
      firstPass = false;
    }
    pos = abs(goalVector);
    percentDone = -((pos/startPos)-1.0)/(0.8);

    if (tcm)
    {
      if ( (percentDone > 0) && (percentDone < 1.0) )
        tcm->completed = percentDone;
      else if ( percentDone < 0 )
        tcm->completed = 0.0;
    }

    if (tm)
      tm->elapsed = ds->currentTime();

//    if (tcm)
//      logPrintf("umbMechanismEvaluator: tcm->completed = %lf\n",tcm->completed);

    if (abs(goalVector) < 0.2)
    {
      if (tcm)
        tcm->completed = 1.0;

      logPrintf("umbMechanismEvaluator:goal reached!\n");
//      taskCompletionMetric::completed;
      done = true;
      applyMetrics(&metrics, this, 0, 1);
//      commandMode = STOP;
//      speed = 0;
//      logPrintf("umbController: At goal.\n");
//      PRINT_TRIPLE(goalVector);
    }
  }

  return retval;
}
```

```
// this adds our special classes to the database so that they can be
// instantiated by classname (e.g. through the p-file)
int initializeUserumbMechanismEvalDB(void) {
  printf("Adding umbMechanism types.\n");
  fflush(stdout);

  // add the evaluator
  ADD_DB_TYPE(d2kSimulator, umbMechanismEvaluator, NULL);

  ADD_DB_TYPE(module, hingeJoint, hingeJoint::initializeDependencyTable);
  ADD_DB_TYPE(module, cubeLink, NULL);
  ADD_DB_TYPE(d2kComponent, umbController, NULL);

  // if we have any specialized d2kComponents, add them
  // ADD_DB_TYPE(d2kComponent, umbMechanismComponent, NULL);
  return 1;
}
// dlsym reads symbols from a dynamic library, but we have to look up
// the symbols as if they were C (not C++).  So, we use this macro to
// make a cover function for initializeUserEvalDB().
MAKE_C_DB_COVER(umbMechanismEval);
```

umbController.H

```
////////////////////////////////////////////////////////////////////////
//
// Darwin2K - simulation and automated synthesis for robotics
// Copyright (C) 2000 P. Chris Leger
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place - Suite 330,
// Boston, MA  02111-1307, USA.
//
// CVS ID:
//   $Id: umbController.h,v 1.15 2002/01/25 23:27:11 xrayjones Exp $
//
// Change history:
//   $Log: umbController.h,v $
//   Revision 1.15  2002/01/25 23:27:11  xrayjones
//   Massive changes - multi-cfg
//
//   Revision 1.14  2001/12/13 18:26:36  xrayjones
//   Adding CVS info to comment blocks
//
//
////////////////////////////////////////////////////////////////////////

#ifndef INCumbController_h
#define INCumbController_h

#ifdef D2K_INTERNAL
# include <jointController.h>
#else
# include <darwin2k/jointController.h>
#endif


/*MAKEDOC++ */

class wheelModule;
class prismaticBeam2M;

/** A controller for wheeled rovers.  The umbController class implements a
 * simple algorithm for skid-steered control of the drive actuators for
 * wheeled rovers.  Several different control modes are available:
 * straight-line, open loop driving; drive-to-location, turn-in-place, and
 * driving along arcs.  The umbController uses the capabilities of the
 * jointController, from which it is derived, to compute joint torque
 * commands for joints connected to wheelModules; all other joint torque
 * commands are zero.  The gains for the singleJointControllers are set
 * to ka, kv, ks, kvt, and kvi, respectively, multiplied by the radius
 * of the wheel being controlled.  This allows control gains for the
 * umbController to be specified in terms of linear velocity and
 * acceleration, regardless of wheel radius.
 * <p>
 * <u>File variables</u>
 * <ul>
 * <li>double accMax <em>required</em>
 * <li>double ka <em>required</em>
 * <li>double kv <em>required</em>
 * <li>double kvt <em>required</em>
 * <li>double kvi = kv*0.1;
```

```
 * <li>enum commandMode { STOP, DRIVE, TURN, ROTATE, SEEK_GOAL } = STOP;
 * <li>double speed = 0;
 * <li>double radius = 0;
 * <li>double kHeading = 20;
 * <li>double goalX = 0;
 * <li>double goalY = 5;
 * <li>int useMassProportionalGains = 0;
 * </ul>
 * Also see <a href="jointController.html">jointController</a> class for other
 * parameters.
 */
class umbController : public jointController {
public:
  /** enumerated type for umbController drive modes.
   * <ul>
   * <li> STOP - decelerate and then hold all wheels at current position.
   * <li> DRIVE - straight-line, open-loop driving at constant speed
   * <li> TURN - arc turn of specified radius
   * <li> ROTATE - turn in place
   * <li> SEEK_GOAL - drive to a specified location
   */

  enum driveMode { STOP = 0, DRIVE, TURN, ROTATE, SEEK_GOAL };

  ptrList wheels;
  ptrList prismaticJoints;
  vector lastCmd;
  vector lastVel;
  vector error;
  vector wheelCmdVel;
  vector jointCmdVel;
  double mass;

  /// linear acceleration to use while driving
  double accMax;

  /// controller gain for computing corrections to reduce heading error
  double kHeading;

  /// linear acceleration gain.
  double ka;

  /// gain for velocity error term
  double kv;

  /// gain for velocity (i.e. additional torque proportional to goal velocity)
  double kvt;

  /// integral gain for accumulated velocity error
  double kvi;

  /// indicates whether gains should be multiplied by rover mass
  int useMassProportionalGains;

  /** a time constant used for computing acceleration during kinematic
   * simulation */
  double tc;

  /// the current mode of the controller
  int commandMode;

  /// the commanded driving speed;  positive = forward, negative = back
  double speed;

  /// turning radius, when in TURN mode;  positive = left, negative = right
  double radius;

  /// goal location for SEEK_GOAL mode
  triple goal;

  /// robot position relative to goal
```

```
  triple goalVector;

  MAKE_COVER_FUNCTIONS(umbController);
  DECLARE_PARENT_CLASS(controller);

  // inherited from d2kComponent
  virtual int readParams(paramParser *p);
  virtual int setVariables(const ptrList *taskParams);

  virtual int init(ptrList *cfgs);
  virtual int computeTorqueCommand(double currentTime,
                  int &computedAcc);
  virtual int computeAccelerationCommand(double currentTime);
  virtual int madeProgress(double currentTime);
  virtual int reset(void);

  /// computes velocity commands for all wheels based on the current mode
  virtual int computeVelocityCommand( void );

  /// returns the i'th wheel
  wheelModule *getWheel(int i) { return (wheelModule *)(wheels.list[i]); }

  /// returns the i'th prismatic joint
  joint *getPrismaticJoint(int i) { return (joint *)(prismaticJoints.list[i]); }
};


#endif /* INCumbController_h */
```

umbController.CXX

```
////////////////////////////////////////////////////////////////////////
//
// Darwin2K - simulation and automated synthesis for robotics
// Copyright (C) 2000 P. Chris Leger
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place - Suite 330,
// Boston, MA  02111-1307, USA.
//
// CVS ID:
//   $Id: umbController.cxx,v 1.23 2002/01/25 23:27:11 xrayjones Exp $
//
// Change history:
//   $Log: umbController.cxx,v $
//   Revision 1.23  2002/01/25 23:27:11  xrayjones
//   Massive changes - multi-cfg
//
//   Revision 1.22  2001/12/13 18:26:36  xrayjones
//   Adding CVS info to comment blocks
//
//
////////////////////////////////////////////////////////////////////////

#include <darwin2k/globals.h>
#include <darwin2k/dynoUser.h>
#include <darwin2k/paramParser.h>
#include <darwin2k/darwin2k.h>
#include <darwin2k/controller.h>
```

```
#include <darwin2k/locomotion.h>
#include <umbController.h>

DEFINE_CLASS_ID(umbController);

static const char *controllerModeNames[] = {
  "STOP", "DRIVE", "TURN", "ROTATE", "SEEK_GOAL"
};
bool engaged = false;
bool frontJointForward = false;
bool frontJointReverse = false;
bool middleJointForward = false;
bool middleJointReverse = false;
bool rearJointForward = false;
bool rearJointReverse = false;
bool setTimeLag = false;
bool initial    = true;
bool firstPass  = true;
double timeLag = 0.0;
double timeLagDelta = 0.0;
double frontMaxGoal = 0.0;
double frontMinGoal = 0.0;
double middleMaxGoal = 0.0;
double middleMinGoal = 0.0;
double rearMaxGoal = 0.0;
double rearMinGoal = 0.0;
double frontMaxTH = 0.0;
double frontMinTH = 0.0;
double middleMaxTH = 0.0;
double middleMinTH = 0.0;
double rearMaxTH = 0.0;
double rearMinTH = 0.0;

//////////////////////////////////////////////////////////////////////
//
// int umbController::init(ptrList *Cfgs)
//
//////////////////////////////////////////////////////////////////////
int umbController::init(ptrList *Cfgs) {
  int i;

  if (!jointController::init(Cfgs)) {
    logError("umbController::init");
    logPrintf("  jointController::init failed\n");
    return 0;
  }

  mass = cfg->computeMass();
  logPrintf("umbController::init mass: %lf\n",mass);
  wheels.clear();
  prismaticJoints.clear();

  // first, find all wheel modules and add to list of wheels
  for (i = 0; i < cfg->numEndPoints(); i++) {
    toolModule *tm = cfg->getEndPoint(i)->tool;
    if (IS_OF_TYPE(tm, wheelModule, 1)) {
      wheels.addPtr(tm);
    }
  }

  // find all prismatic joints and add to list
  for (i = 0; i < cfg->numJoints(); i++)
  {
    joint *jt = cfg->getJoint(i);
    if ( jt->type == joint::PRISMATIC )
      prismaticJoints.addPtr(jt);
  }
  logPrintf("umbController::init prismaticJoints.size: %d\n",prismaticJoints.n);
  for (i=0; i<prismaticJoints.n; i++)
  {
    if ( getPrismaticJoint(i)->limits[0] == 1 )
```

```
    {
      logPrintf("umbController::init prismaticJoints[%d].limits[0] == 1 ",i);
      logPrintf("max:    %lf       min:    %lf\n",    getPrismaticJoint(i)->max(0),
getPrismaticJoint(i)->min(0));
//      logPrintf("max.size: %d \n", getPrismaticJoint(i)->max(0).size() );
    }
  }

  middleMaxGoal = getPrismaticJoint(0)->max(0)-.005;
  middleMinGoal = getPrismaticJoint(0)->min(0)+.005;
//  rearMaxGoal = getPrismaticJoint(1)->max(0)-.005;
  rearMaxGoal = getPrismaticJoint(1)->max(0)-.02;
  rearMinGoal = getPrismaticJoint(1)->min(0)+.005;
//  frontMaxGoal = getPrismaticJoint(2)->max(0)-.005;
  frontMaxGoal = getPrismaticJoint(2)->max(0)-.02;
  frontMinGoal = getPrismaticJoint(2)->min(0)+.005;

  middleMaxTH = middleMaxGoal-.02;
  middleMinTH = middleMinGoal+.02;
  rearMaxTH   = rearMaxGoal-.01;
  rearMinTH   = rearMinGoal+.15;
  frontMaxTH  = frontMaxGoal-.01;
  frontMinTH  = frontMinGoal+.15;
  timeLagDelta = 0.25;
//  timeLagDelta = 0.5;

  logPrintf("umbController::frontMinTH:  %lf    frontMaxTH:  %lf\n",  frontMinTH,
frontMaxTH );

  lastCmd.resize(prismaticJoints.n);
  lastVel.resize(prismaticJoints.n);
  error.resize(prismaticJoints.n);
  wheelCmdVel.resize(prismaticJoints.n);
  jointCmdVel.resize(prismaticJoints.n);

  _maxDt = 0.1;

  reset();

  return 1;
}

////////////////////////////////////////////////////////////////////////
//
// int umbController::reset(void)
//
////////////////////////////////////////////////////////////////////////
int umbController::reset(void) {
  int i;
  int retval = 1;

  retval &= jointController::reset();
  retval &= computeVelocityCommand();

  double mass = cfg->computeMass();

//  drive logic
  engaged = false;
  frontJointForward = false;
  frontJointReverse = false;
  middleJointForward = false;
  middleJointReverse = false;
  rearJointForward = false;
  rearJointReverse = false;
  setTimeLag = false;
  initial    = true;
  firstPass  = true;

  for (i=0; i<prismaticJoints.n; i++)
  {
    singleJointController *sjc
```

```
      = ctrl( getPrismaticJoint(i) );

    sjc->setMode( singleJointController::POSITION, NULL );

    if (useMassProportionalGains)
    {
      sjc->kat = ka*mass;
      sjc->kp = 0.5*kv*kv*mass;
      sjc->kv = kv*mass;
      sjc->ki = sqrt( kv*mass );
      sjc->kvi = kvi*mass;
      sjc->kvt = kvt*mass;
      sjc->maxAcc = accMax;
    }
    else
    {
      sjc->kat = ka;
      sjc->kp = 0.5*kv*kv;
      sjc->kv = kv;
      sjc->ki = sqrt( kv );
      sjc->kvi = kvi;
      sjc->kvt = kvt;
      sjc->maxAcc = accMax;
    }
  }

  for (i=0; i<prismaticJoints.n; i++)
    retval &= ctrl( getPrismaticJoint(i) )->holdPosition();

  logPrintf("called umbController::reset\n");

  return retval;
}

////////////////////////////////////////////////////////////////////////
//
// int umbController::readParams(paramParser *p)
//
////////////////////////////////////////////////////////////////////////
int umbController::readParams(paramParser *p) {
  int retval = 1;
  // set some defaults
  ka = 1.0;
  kv = 0.05;
  tc = 0.2;
  mass = -1;
  useMassProportionalGains = 0;

  retval &= p->GET_DOUBLE(accMax);
  retval &= p->GET_DOUBLE(ka);
  retval &= p->GET_DOUBLE(kv);

  kvi = kv*0.1;
  p->QGET_DOUBLE(kvi);

  retval &= p->GET_DOUBLE(kvt);

  commandMode = STOP;
  speed = radius = 0;
  kHeading = 20.0;

  if (p->QGET_ENUM(commandMode)) {
    if (commandMode < STOP || commandMode > SEEK_GOAL) {
      logError("umbController::readParams");
      logPrintf("  bad commandMode %d\n", commandMode);
      return 0;
    }
  }
  p->QGET_DOUBLE(speed);
  p->QGET_DOUBLE(radius);
  p->QGET_DOUBLE(kHeading);
```

```
  double goalX = 0, goalY = 5.0;
  p->QGET_DOUBLE(goalX);
  p->QGET_DOUBLE(goalY);
  goal = triple(goalX, goalY, 2.0);
  p->QGET_INT(useMassProportionalGains);
  logPrintf("umbController::readparams goalX: %lf  goalY: %lf\n", goalX, goalY);

  return (retval & controller::readParams(p));
}

//////////////////////////////////////////////////////////////////////
//
// int umbController::setVariables(const ptrList *taskParams)
//
//////////////////////////////////////////////////////////////////////
int umbController::setVariables(const ptrList *taskParams) {
  int retval = controller::setVariables(taskParams);

  for (int i = 0; i < taskParams->n; i++) {
    const taskParamRecord *taskParam =
      (const taskParamRecord *)(taskParams->list[i]);
    const param *p = taskParam->p;
    const char *varname = taskParam->varName;

    if (!strcmp(varname, "ka")) {
      ka = p->val;
    } else if (!strcmp(varname, "kv")) {
      kv = p->val;
    } else if (!strcmp(varname, "kvi")) {
      kvi = p->val;
    } else if (!strcmp(varname, "kvt")) {
      kvt = p->val;
    } else if (!strcmp(varname, "tc")) {
      tc = p->val;
    } else if (!strcmp(varname, "accMax")) {
      accMax = p->val;
    } else if (!strcmp(varname, "kHeading")) {
      kHeading = p->val;
      logPrintf("kHeading = %.3f\n", kHeading);
    } else if (!strcmp(varname, "speed")) {
      speed = p->val;
    } else if (!strcmp(varname, "useMassProportionalGains")) {
      useMassProportionalGains = p->ival ? 1 : 0;
    }
  }

  reset();

  return retval;
}

//////////////////////////////////////////////////////////////////////
//
//  int umbController::computeTorqueCommand(double currentTime)
//
//////////////////////////////////////////////////////////////////////
int umbController::computeTorqueCommand(double currentTime,
                                        int &computedAcc) {
  int i, retval;
  computeVelocityCommand();

  if (verboseLevel >= SO_DEBUG) {
    logPrintf("\n---umbController::computeTorqueCommand---\n");
    PRINT_VECTOR(wheelCmdVel);
  }

  retval = jointController::computeTorqueCommand(currentTime, computedAcc);

  return retval;
}
```

```
//////////////////////////////////////////////////////////////////////
//
//  int umbController::computeAccelerationCommand(double currentTime
//                                                double &dt)
//
//////////////////////////////////////////////////////////////////////
int umbController::computeAccelerationCommand(double currentTime){

//  computeVelocityCommand();

  return 1;
}

//////////////////////////////////////////////////////////////////////
//
// int umbController::computeVelocityCommand(void)
//
//////////////////////////////////////////////////////////////////////
int umbController::computeVelocityCommand(void) {
  double phiDot;

  switch (commandMode) {
  case STOP:
    speed = 0;
    phiDot = 0;
    break;
  case DRIVE:
  case SEEK_GOAL:
    phiDot = 0;
    break;
  case TURN:
    phiDot = speed/radius;
    break;
  case ROTATE:
    phiDot = speed;
    radius = 0;
    break;
  }

//  logPrintf("umbController::pos[0]: %lf  pos[2]: %lf  vel[2]: %lf  time: %lf\n",
//     getPrismaticJoint(0)->pos(0), getPrismaticJoint(2)->pos(0),
//     getPrismaticJoint(2)->vel(0), currentTime);

//
// stand up and initialize system
//
  if ( commandMode == DRIVE || commandMode == SEEK_GOAL )
  {
    if (initial)
    {
//      if ( ( fabs(getPrismaticJoint(1)->vel(0)) <= .005 ) &&
//               (  fabs(getPrismaticJoint(2)->vel(0))  <=  .005  )  &&  initial  &&
!setTimeLag && engaged )
      if ((getPrismaticJoint(1)->pos(0) <= rearMinTH) &&
          (getPrismaticJoint(2)->pos(0)  <=  frontMinTH)  &&  initial  &&  !setTimeLag
&& engaged )
      {
        logPrintf("umbController::init::setTimer time: %lf\n", currentTime);
        timeLag = currentTime + timeLagDelta;
        setTimeLag = true;
      }
      if ( currentTime >= 1.0 && !engaged && initial && !firstPass )
      {
        logPrintf("umbController::initialization     started     time:     %lf\n",
currentTime);
        ctrl( getPrismaticJoint(1) )->p = rearMinGoal;
        ctrl( getPrismaticJoint(2) )->p = frontMinGoal;
        engaged = true;
      }
      if ( initial && setTimeLag && (currentTime >= timeLag) )
```

```
      {
        logPrintf("umbController::init::frontJointForward      time:      %lf\n",
currentTime);
        initial = false;
        setTimeLag = false;
        frontJointForward = true;
        ctrl( getPrismaticJoint(2) )->p = frontMaxGoal;
      }
      firstPass = false;
    }
    else
    {
//
//  main movement logic
//
      if ( frontJointForward && !setTimeLag && (getPrismaticJoint(2)->pos(0) >=
frontMaxTH) )
      {
        logPrintf("umbController::setTimer  time: %lf\n", currentTime);
        timeLag = currentTime + timeLagDelta;
        setTimeLag = true;
      }
      if ( frontJointForward && setTimeLag && (currentTime > timeLag) && !initial
)
      {
        logPrintf("umbController::middleJointForward time: %lf\n", currentTime);
        ctrl( getPrismaticJoint(0) )->p = middleMaxGoal;
        frontJointForward  = false;
        middleJointForward = true;
        setTimeLag = false;
      }
      if ( middleJointForward && (!setTimeLag) && (getPrismaticJoint(0)->pos(0) >=
middleMaxTH) )
      {
        logPrintf("umbController::setTimer  time: %lf\n", currentTime);
        timeLag = currentTime + timeLagDelta;
        setTimeLag = true;
      }
      if ( middleJointForward && setTimeLag && (currentTime > timeLag) )
      {
        logPrintf("umbController::frontJointReverse time: %lf\n", currentTime);
        ctrl( getPrismaticJoint(2) )->p = frontMinGoal;
        middleJointForward = false;
        frontJointReverse  = true;
        setTimeLag = false;
      }
      if ( frontJointReverse && (!setTimeLag) && (getPrismaticJoint(2)->pos(0) <=
frontMinTH) )
      {
        logPrintf("umbController::setTimer  time: %lf\n", currentTime);
        timeLag = currentTime + timeLagDelta;
        setTimeLag = true;
      }
      if ( frontJointReverse && setTimeLag && (currentTime > timeLag) )
      {
        logPrintf("umbController::rearJointForward time: %lf\n", currentTime);
        ctrl( getPrismaticJoint(1) )->p = rearMaxGoal;
        frontJointReverse  = false;
        rearJointForward = true;
        setTimeLag = false;
      }
      if ( rearJointForward && (!setTimeLag) && (getPrismaticJoint(1)->pos(0) >=
rearMaxTH) )
      {
        logPrintf("umbController::setTimer  time: %lf\n", currentTime);
        timeLag = currentTime + timeLagDelta;
        setTimeLag = true;
      }
      if ( rearJointForward && setTimeLag && (currentTime > timeLag) )
      {
        logPrintf("umbController::middleJointReverse time: %lf\n", currentTime);
```

```
        ctrl( getPrismaticJoint(0) )->p = middleMinGoal;
        rearJointForward  = false;
        middleJointReverse = true;
        setTimeLag = false;
      }
      if ( middleJointReverse && (!setTimeLag) && (getPrismaticJoint(0)->pos(0) <=
middleMinTH) )
      {
        logPrintf("umbController::setTimer time: %lf\n", currentTime);
        timeLag = currentTime + timeLagDelta;
        setTimeLag = true;
      }
      if ( middleJointReverse && setTimeLag && (currentTime > timeLag) )
      {
        logPrintf("umbController::rearJointReverse time: %lf\n", currentTime);
        ctrl( getPrismaticJoint(1) )->p = rearMinGoal;
        middleJointReverse = false;
        rearJointReverse  = true;
        setTimeLag = false;
      }
      if ( rearJointReverse && (!setTimeLag) && (getPrismaticJoint(1)->pos(0) <=
rearMinTH) )
      {
        logPrintf("umbController::setTimer time: %lf\n", currentTime);
        timeLag = currentTime + timeLagDelta;
        setTimeLag = true;
      }
      if ( rearJointReverse && setTimeLag && (currentTime > timeLag) )
      {
        logPrintf("umbController::frontJointForward  time: %lf\n", currentTime);
        ctrl( getPrismaticJoint(2) )->p = frontMaxGoal;
        frontJointForward = true;
        rearJointReverse  = false;
        setTimeLag = false;
      }
    }
  }
  else if (commandMode == STOP)
  {
    initial = true;
    for (int i=0; i<prismaticJoints.n; i++)
      ctrl( getPrismaticJoint(i) )->holdPosition();
  }

  // create some handy vectors
/*  triple o = cfg->mech->root->T(triple(0));
  triple xb(cfg->mech->root->r(0, 0),
            cfg->mech->root->r(1, 0),
            cfg->mech->root->r(2, 0));
  triple yb(cfg->mech->root->r(0, 1),
            cfg->mech->root->r(1, 1),
            cfg->mech->root->r(2, 1));
  triple zb(cfg->mech->root->r(0, 2),
            cfg->mech->root->r(1, 2),
            cfg->mech->root->r(2, 2));

  goalVector = goal-o;
//  logPrintf("umbController: ");
//  PRINT_TRIPLE(goalVector);

  if (abs(goalVector) < 0.2)
  {
    speed = 0;
    logPrintf("umbController: At goal.\n");
    PRINT_TRIPLE(goalVector);
  }
*/
  if (verboseLevel >= SO_DEBUG) {
    logPrintf("---umbController::computeVelocityCommand()---\n");
    logPrintf("umbController: mode = %s\n",
              controllerModeNames[commandMode]);
```

```
    logPrintf("  speed: %.3f  radius: %.3f  phiDot: %.3f\n",
              speed, radius, phiDot);
//    PRINT_TRIPLE(o);
//    PRINT_TRIPLE(xb);
//    PRINT_TRIPLE(yb);
//    PRINT_TRIPLE(zb);
  }

  if (commandMode == TURN || commandMode == ROTATE) {
/*    triple o(mech->root->T(0));
    triple c = o - radius*xb;

    if (verboseLevel >= SO_DEBUG) PRINT_TRIPLE(c);

    for (int i = 0; i < wheels.n; i++) {
      wheelModule *wm = getWheel(i);
      if (!wm->getJoint() || !wm->winfo) continue;
      wm->winfo->update();
      triple xwheel = wm->winfo->center;
      triple w(xwheel - c);
      triple vw((w ^ zb)*phiDot);
      triple d = vw - (vw*wm->axis())*wm->axis();
      double v = abs(d);
      d = d/v;

      if (wm->axis() * d > 0) wheelCmdVel(i) = -v/wm->radius;
      else wheelCmdVel(i) = v/wm->radius;

      if (verboseLevel >= SO_DEBUG) {
        PRINT_TRIPLE(w);
        PRINT_TRIPLE(vw);
        PRINT_TRIPLE(d);
        logPrintf("v: %.3f\n", v);
      }
    } */
  } else {
/*    double headingError = 0;
    if (commandMode == SEEK_GOAL) {
      triple goalVector = goal-o;
      if (abs(goalVector) < 0.5) {
        commandMode = STOP;
        speed = 0;
        if (verboseLevel >= SO_INFO) {
          logPrintf("umbController: At goal.\n");
        }
      }
      goalVector(2) = 0;
      goalVector.normalize();
      xb(2) = 0;
      xb.normalize();
      headingError = goalVector*xb;
      if (headingError > 0.999) headingError = M_PI/2;
      else if (headingError < -0.999) headingError = -M_PI/2;
      else headingError = asin(headingError);
      if (fabs(headingError) < 0.005) headingError = 0;

      if (verboseLevel >= SO_INFO) {
        logPrintf("headingError: %g\n", headingError);
      }
    }
    if (verboseLevel >= SO_DEBUG) {
      PRINT_TRIPLE(yb);
    }
/*
    for (int i = 0; i < wheels.n; i++) {
      wheelModule *wm = getWheel(i);
      if (!wm->getJoint() || !wm->winfo) continue;
      wm->winfo->update();
      triple xwheel = wm->winfo->center;
      double dp = (wm->axis() ^ yb)*zb;
      double wheelDist = (xwheel-o)*xb;
```

```
      double wheelSpeed;

      if (fabs(wheelDist) > wm->radius/2 && headingError > 0.005 &&
          wheelDist*headingError < 0) {
        // slow this wheel down
        double ratio = 1-headingError*kHeading;
        if (ratio < -0.5) ratio = -0.5;
        wheelSpeed = (speed/wm->radius) *
          ratio * wheelDist/fabs(wheelDist);
        if (wheelSpeed < 0) wheelSpeed = 0;
      } else {
        wheelSpeed = speed/wm->radius;
      }

      if (verboseLevel >= SO_DEBUG) {
        PRINT_TRIPLE(xwheel);
        logPrintf("wheelDist: %g\n", wheelDist);
        logPrintf("wheelSpeed: %g\n", wheelSpeed);
      }

      if (dp > 0) {
        wheelCmdVel(i) = -wheelSpeed;
        if (verboseLevel >= SO_DEBUG) logPrintf("-speed/radius: %.3f\n",
                                                wheelCmdVel(i));
      } else {
        wheelCmdVel(i) = wheelSpeed;
        if (verboseLevel >= SO_DEBUG) logPrintf("+speed/radius: %.3f\n",
                                                wheelCmdVel(i));
      }
    } */
  }
  return 1;
}

////////////////////////////////////////////////////////////////////////
//
//  int umbController::madeProgress(double currentTime)
//
////////////////////////////////////////////////////////////////////////
int umbController::madeProgress(double currentTime) {
  return 1;
}
```

APPENDIX B

Darwin2K / UmbrellaBot Input Files

UMB_EV_3.L – Evaluation Configuration File

```
((hexapodBase ((const 0.15 0.15 2 0)
               (const 0.15 0.15 2 0)
               (const 0.15 0.15 2 0)
               (const 0.001 0.001 2 0))
              ((const 4 (1 0 left  (const 0 270 2 2)))
               (const 5 (2 0 right  (const 0 270 2 2)))
 ))
 (prismaticBeam2M lowInertia ((const 0 1 3 2)
      (const 0 1 5 10)
      (const 0 1 2 2)
      (const 0 1 2 0)
      (const 0.02 0.15 3 3)
      (const 0.001 0.005 3 0)
      (var 1.0 1.5 3 6))
   ((const 1 (14 0 inherit (const 0 270 2 0)))))
 (cubeLink hollowTube ((const 0 1 2 0)
      (const 0.005 0.08 3 7)
      (const 0.005 0.08 3 7)
      (const 0.0015 0.005 3 7))
   ((const 1 (5 0 inherit (const 0 270 2 0)))
    (const 2 (3 0 inherit (const 0 270 2 0)))
    (const 3 (3 0 inherit (const 0 270 2 0)))
    (const 4 (3 0 inherit (const 0 270 2 0)))
    (const 5 (3 0 inherit (const 0 270 2 0)))
   ))
 (hingeJoint freeRevoluteJoint ((const 0 1 2 2)
      (const 0 1 2 10)
      (const 0 1 2 0)
      (const -1.57 1.57 6 52)
      (const 0.005 0.015 3 3)
      (const 0 0.2 2 0)
      (const .01 .01 2 0)
      (const .05 .05 2 0))
   ((const 1 (4 0 inherit (const 0 270 2 0)))
   ))
 (hollowTube hollowTube ((const 0 1 2 0)
      (const 0.55 0.55 2 0)
      (const 0.02 0.15 3 3)
      (const 0.001 0.005 3 0))
   ((const 1 (11 0 inherit (const 0 270 2 0)))
   ))
 (prismaticBeam2M lowInertia ((const 0 1 3 2)
      (const 0 1 5 10)
      (const 0 1 2 2)
      (const 0 1 2 0)
      (const 0.02 0.15 3 3)
      (const 0.001 0.005 3 0)
      (const 0.05 0.7 3 6))
   ((const 1 (6 1 inherit (const 0 270 2 0)))
   ))
 (cubeLink hollowTube ((const 0 1 2 0)
      (const 0.005 0.08 3 7)
      (const 0.005 0.08 3 7)
      (const 0.0015 0.005 3 7))
    ((const 2 (7 0 inherit (const 0 270 2 0)))
     (const 3 (7 0 inherit (const 0 270 2 0)))
     (const 4 (7 0 inherit (const 0 270 2 0)))
     (const 5 (7 0 inherit (const 0 270 2 0)))
    ))
 (hingeJoint freeRevoluteJoint ((const 0 1 2 2)
      (const 0 1 2 10)
      (const 0 1 2 0)
      (const -1.57 1.57 6 41)
      (const 0.005 0.015 3 3)
      (const 0 0.2 2 0)
      (const .01 .01 2 0)
      (const .05 .05 2 0))
   ((const 1 (8 0 inherit (const 0 270 2 0)))
   ))
 (hollowTube hollowTube ((const 0 1 2 0)
      (const 0.39 0.39 2 0)
      (const 0.02 0.15 3 3)
      (const 0.001 0.005 3 0))
   ((const 1 (9 0 inherit (const 0 270 2 0)))
   ))
 (cubeLink hollowTube ((const 0 1 2 0)
      (const 0.005 0.08 3 7)
      (const 0.005 0.08 3 7)
      (const 0.0015 0.005 3 7))
   ((const 1 (12 0 inherit (const 0 270 2 0)))
    (const 4 (11 0 inherit (const 0 270 2 0)))
   ))
 (hingeJoint freeRevoluteJoint ((const 0 1 2 2)
      (const 33 33 6 63)
      (const 0 1 2 0)
      (const -1.57 1.57 6 20)
      (const 0.005 0.015 3 3)
      (const 0 0.2 2 0)
      (const .01 .01 2 0)
      (const .05 .05 2 0))
  ((const 1 (11 0 inherit (const 0 270 2 0)))
 ))
 (simpleTool ((const 0.05 0.1 3 7)) nil)
 (hollowTube hollowTube ((const 0 1 2 0)
      (var 1.405 1.415 2 0)
      (const 0.02 0.15 3 3)
      (const 0.001 0.005 3 0))
   ((const 1 (13 0 inherit (const 0 270 2 0)))
   ))
 (rubberFoot ((const 0.01 0.6 4 4)
      (const 0.005 0.08 4 15)
      (const 0.0015 0.005 3 7)
      (const 500 4000 4 12)
      (const 10 500 4 0))
   nil)
 (cubeLink hollowTube ((const 0 1 2 0)
      (const 0.005 0.08 3 7)
      (const 0.005 0.08 3 7)
      (const 0.0015 0.005 3 7))
   ((const 1 (17 0 inherit (const 0 270 2 0)))
    (const 2 (15 0 inherit (const 0 270 2 0)))
    (const 3 (15 0 inherit (const 0 270 2 0)))
    (const 4 (15 0 inherit (const 0 270 2 0)))
    (const 5 (15 0 inherit (const 0 270 2 0)))
   ))
 (hingeJoint freeRevoluteJoint ((const 0 1 2 2)
      (const 0 1 2 10)
      (const 0 1 2 0)
      (const -1.57 1.57 6 11)
      (const 0.005 0.015 3 3)
      (const 0 0.2 2 0)
      (const .01 .01 2 0)
      (const .05 .05 2 0))
   ((const 1 (16 0 inherit (const 0 270 2 0)))
   ))
 (hollowTube hollowTube ((const 0 1 2 0)
      (const 0.55 0.55 2 0)
      (const 0.02 0.15 3 3)
      (const 0.001 0.005 3 0))
   ((const 1 (23 0 inherit (const 0 270 2 0)))
   ))
 (prismaticBeam2M lowInertia ((const 0 1 3 2)
      (const 0 1 5 10)
      (const 0 1 2 2)
      (const 0 1 2 0)
      (const 0.02 0.15 3 3)
      (const 0.001 0.005 3 0)
      (const 0.05 0.7 3 6))
    ((const 1 (18 0 inherit (const 0 270 2 0)))
    ))
 (cubeLink hollowTube ((const 0 1 2 0)
      (const 0.005 0.08 3 7)
      (const 0.005 0.08 3 7)
      (const 0.0015 0.005 3 7))
    ((const 2 (19 0 inherit (const 0 270 2 0)))
     (const 3 (19 0 inherit (const 0 270 2 0)))
     (const 4 (19 0 inherit (const 0 270 2 0)))
     (const 5 (19 0 inherit (const 0 270 2 0)))
    ))
 (hingeJoint freeRevoluteJoint ((const 0 1 2 2)
      (const 0 1 2 10)
      (const 0 1 2 0)
      (const -1.57 1.57 6 41)
      (const 0.005 0.015 3 3)
      (const 0 0.2 2 0)
      (const .01 .01 2 0)
      (const .05 .05 2 0))
   ((const 1 (20 0 inherit (const 0 270 2 0)))
   ))
 (hollowTube hollowTube ((const 0 1 2 0)
      (const 0.39 0.39 2 0)
      (const 0.02 0.15 3 3)
      (const 0.001 0.005 3 0))
   ((const 1 (21 0 inherit (const 0 270 2 0)))
   ))
 (cubeLink hollowTube ((const 0 1 2 0)
      (const 0.005 0.08 3 7)
      (const 0.005 0.08 3 7)
      (const 0.0015 0.005 3 7))
   ((const 1 (24 0 inherit (const 0 270 2 0)))
    (const 5 (23 0 inherit (const 0 270 2 0)))
   ))
 (hingeJoint freeRevoluteJoint ((const 0 1 2 2)
      (const 33 33 6 63)
      (const 0 1 2 0)
      (const -1.57 1.57 6 13)
      (const 0.005 0.015 3 3)
      (const 0 0.2 2 0)
      (const .01 .01 2 0)
      (const .05 .05 2 0))
  ((const 1 (23 0 inherit (const 0 270 2 0)))
 ))
 (simpleTool ((const 0.05 0.1 3 7)) nil)
 (hollowTube hollowTube ((const 0 1 2 0)
      (var 1.405 1.415 2 0)
      (const 0.02 0.15 3 3)
      (const 0.001 0.005 3 0))
   ((const 1 (25 0 inherit (const 0 270 2 0)))
   ))
 (rubberFoot ((const 0.01 0.6 4 4)
      (const 0.005 0.08 4 15)
      (const 0.0015 0.005 3 7)
      (const 500 4000 4 12)
      (const 10 500 4 0))
   nil)
```

UMBEVALPM.P– Evaluation Parameter File

```
char evalType[80] = "umbMechanismEvaluator";
char componentDBFilename[80] = "componentDB";
int recordFrames = 0;
char animBaseName[80] = "/export/samba/linux_share/movies/f";
int addJointLimitConstraints = 0;
int numTrials = 3;

int lowDetailTubeNumSides = 6;

int useKinematicFriction = 0;
int addDynamicJointFriction = 0;
int addStaticJointFriction = 0;

int componentVerboseLevel = 1;
int constraintVerboseLevel = 0;
int dynamicVerboseLevel = 1;
int cfgVerboseLevel = 1;
int patchVerboseLevel = 1;

double goalX = 1.0;
double goalY = 0.0;

#libraries
char plugin0[80] = "umbMechanism";
char auxFunc0[80] = "glDisplayInit";

#cfg0
double initPosX = 0;
double initPosY = -0.1;
double initPosZ = 2.0;
double initQR = 0.9;
double initQI = -0.4;
double initQJ = 0;
double initQK = 0;

/*
double initQR = 0.9063;
double initQI = -0.4226;
double initQJ = 0;
double initQK = 0; */

#dynamicSystem
/* int ticksPerSecond = 500; */
char solverContext[100] = "rungeKutta4";

#rungeKutta4
double maxDt = 0.005;
double minDt = 1e-5;
double tol = 1e-5;
int maxSteps = 20;
int verboseLevel = 1;

#synGLDisplay
int verboseLevel = 1;
int useLowDetail = 1;
int smoothShading = 1;

float backgroundR = 1.0;
float backgroundG = 1.0;
float backgroundB = 1.0;

double eyePosX = 6.0;
double eyePosY = -1.0;
double eyePosZ = 4.0;
double lookAtX = 2.0;
double lookAtY = 0.0;
double lookAtZ = 0.7;
/*
double eyePosX = -5.0;
double eyePosY = 7.0;
double eyePosZ = 4.0;
double lookAtX = 0.0;
double lookAtY = 3.0;
double lookAtZ = 0.7; */

double vupX = 0.0;
double vupY = 0.0;
double vupZ = 1.0;
double fov = 70.0;

#evaluator
double endTime = 10.0;
int useDisplay = 1;
int verboseLevel = 0;
double maxRealtime = 1000.0;
double simTimeout = 10.0;
int startPaused = 1;
double dt = 0.01;
/*double dt = 0.002;*/
/*
** gravity vector given in m/s**2
*/
double gravityX = 0.0;
double gravityY = 0.0;
double gravityZ = -2.5;
/*double gravityZ = -9.81;*/

#umbMechanismEvaluator
int numConstraints = 8;
char pointFile[80] = "points.dat";

#component0
char class[80] = "umbController";
char label[80] = "controller";
int enabled = 1;
```

```
double maxDt = 0.01;
int alwaysUpdate = 1;
int useMassProportionalGains = 1;
double ka = 5.0;
double kv = 20.0;
int verboseLevel = 1;
double ks = 0.0;
double kvi = 1.0;
double kvt = 0.0;
enum commandMode { STOP, DRIVE, TURN, ROTATE, SEEK_GOAL } = SEEK_GOAL;
double accMax = 1.0;
double speed = 0.1;
double radius = 0.0;
double goalX = 2.5;
double goalY = 0.0;
/*double goalY = (14+1.0);*/
double kHeading = 40.0;

#component1
char class[80] = "terrainModel";
char label[80] = "terrain";
int verboseLevel = 1;
int enabled = 1;
double kr = 0;
double kp = -2000;
/* double muS = 0.7; */
/* double muD = 0.6; */
double muS = 0.9;
double muD = 0.9;
double tol = 0.005;
char terrainFilename[80] = "cyl3.dat";

#component2
char class[80] = "glDisplayHook";
char label[80] = "masterDisplay";
int startPaused = 1;
int enabled = 0;
```

PM.P– Performance Metric Parameter File

```
#populationManager

enum selectionMode { weightedSum, cdf, mdf, req } = req;
char evalType[80] = "umbMechanismEvaluator";
char evalParamFile[200] = "umbEvalPM.p";
char componentDBFilename[80] = "componentDB";
char kernelFilename[80] = "umb_ev_3.1";

int numTaskParams = 0;
int newMetricSpec = 1;
int numMetrics = 5;

int initPopSize = 5;
int creationTimeout = 2000;
int popSize = 10;
int maxIndividuals = 100;
int stageLimit = 400;
int scaleMetrics = 0;
int stopAtMax = 1;
int logPopulation = 1;
double cullRate = 0.0;
double cullFraction = 0.000;
double insertionRate = 0;
double replacementRate = 0;
double deletionRate = 0;
double permutationRate = 0;
double cpXoverRate = 0;
double xoverRate = 0;
double paramXoverRate = 0.95;
double paramMutationRate = 0.05;
int readPopulation = 0;
double runTime = 10.0;
#libraries
char plugin0[80] = "umbMechanism";

/*********************************************************************
 * metric info
 *********************************************************************/

/*********************************************************/
#metric0
char name[80] = "taskCompletionMetric";
enum mode {MIN, MAX, AVG, RMS, INTEGRAL} = MAX;
/* enum mode {MIN, MAX, AVG, RMS, INTEGRAL} = MIN; */
double wt = 1000.0;
double min = 0.0;
double max = 1.0;
double scale = 0.1;
/* double scale = 1; */
int priority = 0;
char op[20] = "==";
double thresh = 1.0;
/* double thresh = 1.0; */

/*********************************************************/
#metric1
char name[80] = "massMetric";
enum mode {MIN, MAX, AVG, RMS, INTEGRAL} = MAX;
double wt = 1.0;
double min = 0.1;
double max = 300.0;
double afdi = 1;  /* 1kg */
int priority = 1;
char op[20] = "NOP";
double thresh = 0.0;

/*********************************************************/
#metric2
char name[80] = "timeMetric";
enum mode {MIN, MAX, AVG, RMS, INTEGRAL} = MAX;
double wt = 1.0;
double min = 0.0;
double max = 100.0;
double afdi = 1;
int priority = 1;
char op[20] = "NOP";
double thresh = 0.0;

/*********************************************************/
#metric3
char name[80] = "powerMetric";
enum mode {MIN, MAX, AVG, RMS, INTEGRAL} = INTEGRAL;
double wt = 1.0;
double min = 0.0;
double max = 20000.0;
double scale = 0.007;   /* 100J less = 2x selection prob */
int priority = 2;
char op[20] = "NOP";
double thresh = 0.0;

/*********************************************************/
#metric4
char name[80] = "linkDeflectionMetric";
enum mode {MIN, MAX, AVG, RMS, INTEGRAL} = MAX;
double wt = 1.0;
double min = 0.0009;
double max = 0.01;
double scale = 693.0;  /* 1mm less = 2x prob */
int priority = 2;
char op[20] = "<=";
double thresh = 0.001;
```

CYL3.DAT– Terrain Definition File

```
1
polyhedron
0.8 0.6
1
0.7  0.5  0.2
16

-5.00000        0.82843         0.00000
-5.00000        2.00000         1.17157
-5.00000        2.00000         2.82843
-5.00000        0.82843         4.00000
-5.00000        -0.82843        4.00000
-5.00000        -2.00000        2.82843
-5.00000        -2.00000        1.17157
-5.00000        -0.82843        0.00000
7.00000         0.82843         0.00000
7.00000         2.00000         1.17157
7.00000         2.00000         2.82843
7.00000         0.82843         4.00000
7.00000         -0.82843        4.00000
7.00000         -2.00000        2.82843
7.00000         -2.00000        1.17157
7.00000         -0.82843        0.00000

7

4
1 0 8 9
4
2 1 9 10
4
3 2 10 11
4
4 3 11 12
4
5 4 12 13
4
7 6 14 15
4
0 7 15 8
```