Re-usable Kinematic Models and Algorithms for Manipulators and Vehicles

Hari D. Nayar, Issa A.D. Nesnas
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, M/S 82-105 Pasadena CA 91001, USA
Hari.D.Nayar@jpl.nasa.gov, Issa.A.Nesnas@jpl.nasa.gov

***Abstract*— A generalized kinematic modeling framework, called Mechanism_Model, has been developed for use in the CLARAty robotic reusable software. Mechanism_Model supports a wide range of systems – from manipulator arms to legged and wheeled rovers. It also enables the development of generalized kinematics, dynamics and collision detection algorithms. In this paper, we describe the unified modeling approach used in Mechanism_Model and provide details of its object-oriented implementation in C++. We also present an example application illustrating use of Mechanism_Model.**

I. INTRODUCTION

IN this paper, we describe progress we have made [5] toward a unified modeling and control approach for a wide variety of robotics systems. We report on implementation details of the modeling software package and applications that demonstrate its use. The software, called the *Mechanism_Model* package is being developed within the CLARAty software system [17], [9], [10], [3]. CLARAty is a collaborative effort among four institutions: Jet Propulsion Laboratory, NASA Ames Research Center, Carnegie Mellon, and the University of Minnesota. In this section, we first motivate the development of *Mechanism_Model*, describe its background and discuss its relevance with respect to related work. We then describe, in Section II, the application domains that *Mechanism_Model* is designed to span, the features of these domains and the challenges they pose in developing a unified modeling approach. In Section III, we present the modeling framework used in *Mechanism_Model*, its generic kinematic algorithms and its object-oriented implementation. An example application that demonstrates the usage of *Mechanism_Model* is covered in Section IV. We conclude with a summary of our development and a description of future plans to incorporate it into a generalized control paradigm.

A. CLARAty Software

CLARAty is a framework for reusable robotic software. In an object-oriented hierarchy, at its lowest level, CLARAty implements software abstractions for hardware interfaces. Upon this hardware abstraction layer, re-usable software components are built to interface to higher levels of control. As a result, software that implements complex behaviour and sophisticated operations is platform independent. Examples of such capabilities implemented in CLARAty include pose estimation, navigation, locomotion and planning. In addition to supporting multiple algorithms, CLARAty provides adaptations to multiple robotic and rover platforms. CLARAty is a domain-specific robotic architecture designed with four main objectives:

1. To promote the reuse of robotic software infrastructure across multiple research efforts
2. To promote the integration of new technologies developed by the robotics community onto rover platforms
3. To mature robotic capabilities through reuse and enable independent formal validation
4. To share the development with the robotic community to promote rapid advancement and leveraging of capabilities

The infrastructure to support these objectives has been developed over several years. This project uses an iterative development process that captures lessons learned from the deployment of earlier versions of the framework on real and simulated platforms. Through this process, both the design of the interfaces and the implementation of generic capabilities mature over time. Many elements of CLARAty are in their third revisions with improved interfaces to actuators and sensors, camera modelling and image processing, mechanism modelling, pose estimation, navigation and interfaces to higher level planners. We describe our efforts in developing CLARAty's next generation mechanism modelling software in this paper.

B. Motivation and Objectives

The motivation for the development of *Mechanism_Model* began with the original need in CLARAty to develop kinematic and control algorithms that could be applied with minimal change to the range of robotic vehicles used at JPL. This goal was successfully accomplished with the *Wheel_Locomotor* module in CLARAty. *Wheel_Locomotor* consists of a set of classes that plan and execute circular drive paths for vehicles that are fully-steered, partially-steered or non-steered (skid-steered) with any number of wheels. The early implementations in CLARAty had a separate set of model representations and algorithms for the control of manipulator arms on the rovers from those used for the control of the vehicle. However, the similarity between modeling and control needs for vehicles and the manipulator arms led our desire to unify the models and algorithms. Based on the premise that the more generic the software, the more re-usable it will be, we have attempted to

develop a unified framework for handling kinematics and control of all articulation on our robotic platforms.

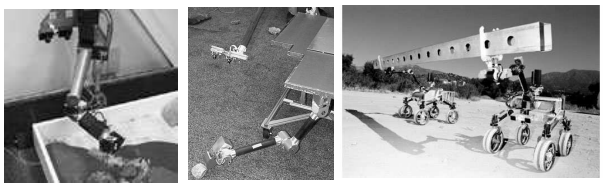
There are a number of other advantages to be gained from a unified approach for modeling and control for mobility systems. One key benefit to a unified representation is the ability to develop more sophisticated algorithms that treat appendages as an extension of the mobility enabling greater flexibility for manipulating arm/vehicle workspaces. Although requiring greater effort to develop, a unified implementation results in less management overhead. In a software system as large and complex as CLARAty, with a variety of application platforms, a streamlined system for handling model data and implementation of algorithms that automatically configure to the platform can simplify system operation and increase its robustness. Additionally, this approach:

- Provides centralized storage for managing model information. This includes creation, deletion, update, extension and reconfiguration of the mechanical models.
- Ensures consistency of the model information for use by multiple algorithms. This simplifies the integration of algorithms into the software architecture. A significant benefit from this approach is that generalized algorithms can then be written for these systems because all mechanisms share a common data structure.
- Reduces duplication in model representation between rover mobility and manipulation software leading to reduced model management overhead.
- Enables the development of generic algorithms for forward, inverse, and differential kinematics. In the absence of specialized versions, the generic algorithms provide out-of-the-box functionality.
- Supports specific implementations to override generic algorithms whenever appropriate for optimal performance.
- Enables the verification of specialized kinematics algorithms against their generic counterpart

C. Related Work

Mechanism_Model uses a kinematics and dynamics modelling structure similar to the one used in the DARTS/Dshell development at JPL [6], [16], [4]. DARTS/Dshell and its derived simulators have been used to simulate, with high-fidelity, the dynamics of robotic systems and spacecraft modeled as flexible mullti-body systems. It has also been used for component hardware-in-the-loop testing, pre-deployment guidance and control algorithm testing and in stand-alone simulations.

The Kinematics and Dynamics Library (KDL) component in the recent software release of Version 1.0 from the Open RObot COntrol System [13] project has objectives similar to our *Mechanism_Model* development. The current release, KDL 0.2.1, implements many useful utilities for kinematic computations needed in manipulator control. The highest level robot model supported in KDL at this time is a serial-link manipulator. The design approach for the modelling mechanisms in KDL is different from the approach we have taken with *Mechanism_Model*. While *Mechanism_Model* breaks down a mechanism model into component bodies and multiple DOF joints, KDL models a serial-link chain as composed of single DOF joints. KDL does not currently have the capability for modelling mobility systems like legged and wheeled rovers.



a) b) c)

Figure 1.Manipulator examples: a) Modular arm, b) Manipulator-mounted cameras guiding instrument placement and c) Rovers cooperatively transporting a beam.

The Operational Software Components for Advanced Robotics [14], [7], developed by the Robotics Research Group at the University of Texas at Austin, provides utilities in the form of libraries for performing computations needed in analysis, real-time control, and simulation of manipulators. In addition to math utilities, it contains algorithms for performing generic forward and inverse kinematics, motion planning and dynamics. OSCAR offers many alternative options in its operations. For example, for motion planning, trajectories can be generated using trapezoidal, spline or motion blending algorithms. OSCAR currently appears to allow only the modeling of serial-chain manipulators. OSCAR's primary application is robotics education. While OSCAR provides generic software utilities for robot arms (serial-chain manipulators) the approach with *Mechanism_Model* models more general kinematics systems.

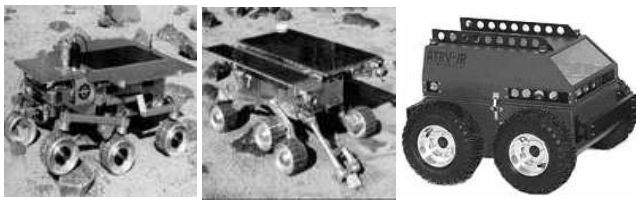
Other related developments include the RoboML [15], ORCA [12], [2] and the Nucleus robotic control toolkit [11].

II. Application Domains and Goals

The wide range of systems to be addressed with our approach present a number of challenges. In the following sections, we categorize the types of systems *Mechanism_Model* will support and discuss their unique features.

A. Manipulators

Serial-link manipulators are the simplest mechanisms we will model in *Mechanism_Model*. The range of mechanisms on rover platforms that can be modeled as serial-link manipulators include passively set or actively controlled pan-and-tilt units for cameras, two degrees-of-freedom (DOF), three or four DOF masts arms mounted with navigation camera heads and four, five, or six DOF instrument arms. These mechanisms are all mounted to mobile rover platforms. Instrument arms have the additional kinematic feature of typically holding many instruments on a turret at its end. In research and flight projects at JPL, serial-link manipulators are also deployed from non-mobile lander platforms. For example, in addition to a scoop as its end effector, the four DOF Mars Phoenix Mars manipulator has a camera attached to its lower arm. It is also desirable that



a) b) c)

Figure 2. Rover examples: a) Rocky 8, b) Rocky 7 and c) ARTV Jr.

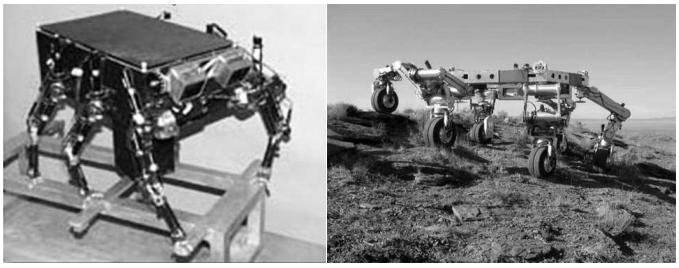
Mechanism_Model be able to also handle parallel-link (or closed-chain) kinematic structures. Examples of manipulators used on research projects at JPL are shown on Figure 1.

B. Wheeled Rovers

Most wheeled mobility systems for research and flight missions at JPL have six wheels and use the passive rocker-bogie suspension [1]. The rocker-bogie mechanism enforces a coupling between the left and right sides in the rocker articulation. This may be modeled as a single independent joint and a dependent (or constrained) joint with respect to the rover chassis. Rover wheels may be steered or non-steered. Rocky 7 rover has two steerable wheels. The Mars Exploration Rovers (MER) have four steerable wheels while Rocky 8 and FIDO have six steerable wheels. The K-10 rover at the NASA Ames Research Center has a rocker mechanism with four steerable wheels and no bogie. At JPL, we also use commercial-off-the-shelf (COTS) mobility platforms for research applications. These are typically four non-steerable wheeled mobility systems with no suspension mechanism. Vehicle steering is accomplished using skid-steering with wheels on opposite sides differentially driven to change vehicle heading. Another feature of some of these skid-steered rovers is that one controlled motor is used to drive both wheels on one side through a belt transmission. Figure 2 shows examples of rovers CLARAty has been implemented on.

C. Limbed Mobility Systems

Another class of mobility systems at JPL is the limbed



a) b)

Figure 3. Hybrid and limbed system examples: a) LEMUR robot, and b)ATHLETE rover.

locomotor. Examples of robots in this category are JPL's LEMUR (Limbed Excursion Mechanical Utility Robots) robots [8]. These are four or six limbed systems that can walk and climb. The first version of LEMUR has six limbs. Its two front limbs have a three-fingered gripper, allowing their use as arms. More recent versions of Lemur have limbs with interchangeable tools and instruments. The first prototype of LEMUR is shown on Figure 3a.

D. Hybrid Systems – Wheeled Legs

The final class of mechanisms we consider is hybrid mobility systems that drive on wheels but may also walk on limbs or use limbs as manipulators. The ATHLETE (All-Terrain Hex-Limbed Extra-Terrestrial Explorer) rover at JPL [18] has six legs each with six DOF. The legs are mounted symmetrically around a hexagonal base. Each leg has a wheel at its tip. With its wheel locked, ATHLETE can walk using its legs. And on relatively flat terrain, to conserve power, ATHLETE can drive on its wheels while using its legs as an active suspension system. Figure 3b shows the ATHLETE rover climbing a hill. As NASA develops innovative mobility systems for exploring steeper and rougher terrain, we will see new hybrid mobility systems that combine multiple modes of locomotion.

III. Modeling and Algorithmic Framework

A. Design Requirements

The primary requirement for the development of *Mechanism_Model* is a unified modeling data structure for the variety of mechanisms to allow the interoperability of models and algorithms. Separation of the kinematics and dynamics data and algorithms from control software is also to be enforced so that model-related algorithms can be run independent of physical systems. For greater efficiency in real-time control applications, users should have the option to override generic algorithms and use customized algorithms for specific systems. Customized algorithms, however, will use parameters from the common unified model. Algorithms to be included in *Mechanism_Model* are forward and inverse position, and velocity kinematics, quasi-static computations of forces and torques that include models of joint flexibility, gravity force and other applied forces, gravity deflection, environmental contact constraints and collision detection.

B. Model Framework

Using the approach in DARTS/Dshell (Rodriguez, 1991, Jain 1991), *Mechanism_Model*, models elements of mechanisms as bodies arranged in a tree structure as illustrated on Figure 4. *Mechanism_Model* bodies represent the rigid or flexible components of a mechanism that articulate with respect to each other. Articulations are modeled as joints. A ground body, representing the inertial frame in the system, is at the base of the tree. Component bodies have *only one parent* but may have many child bodies. The ground body is the only body without a parent.

Bodies are rigid in the current implementation of *Mechanism*_Model but may be extended to be flexible. Bodies have required kinematic attributes for kinematic algorithms and optional inertial, geometric, and visual attributes for use in dynamics and collision detection algorithms and for graphics display.

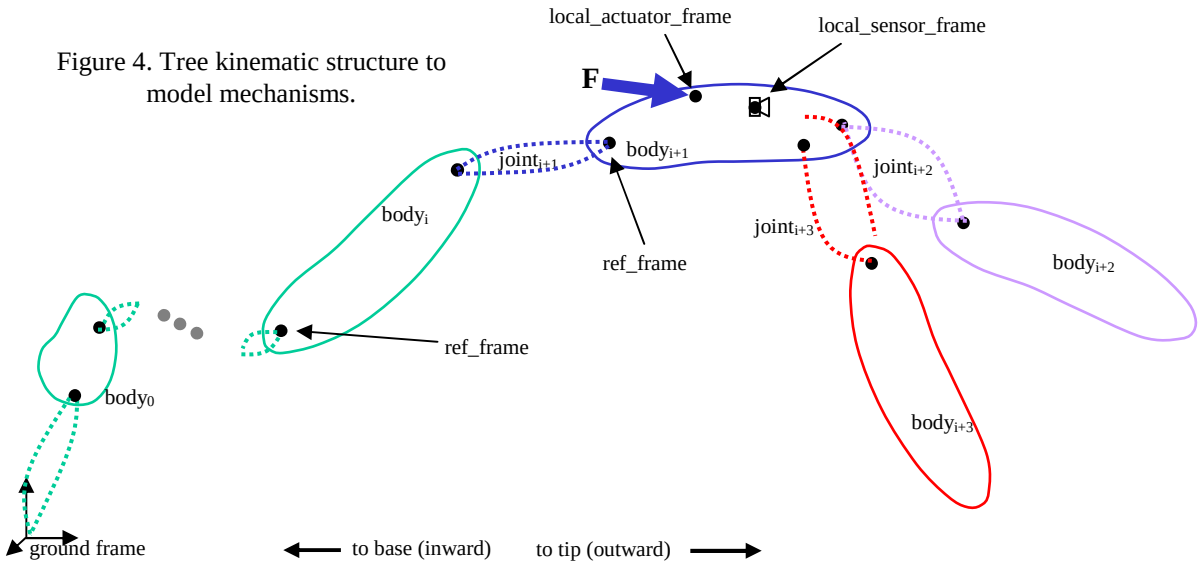


Figure 4. Tree kinematic structure to model mechanisms.

Bodies are connected to each other by joints. Joint articulation between pairs of bodies occurs between an output frame of the parent body and the reference frame of the child body. A joint can have multiple degrees of freedom, may be actively controlled or passive, and may be constrained to have its articulation state depend on the state of another previously defined joint. An example of the use of a constrained joint is in modeling the rocker-bogie differential with opposite sides having opposite angles. Joints have type (1 DOF prismatic, 1 DOF revolute, 6DOF spatial, etc.), offset value (zero joint position offset from the joint coordinate frame) and home position (joint position at robot home position) attributes Joints also have optional articulation limits, stiffness and constraint attributes. The kinematic relationship between a parent body and a child body is specified by the type of joint, its attachment on the parent body and the joint articulation state.

In *Mechanism_Model*, frame objects are used to represent coordinate frames of interest on bodies. There are two types of frame objects: reference frame objects and local frame objects. Each body has one reference frame object and may have multiple local frame objects. In addition to its type, a frame object has the attributes of a homogenous transform and a string label. A body's reference frame object is used to specify its nominal (zero joint values) pose with respect to its parent. Local frame objects on a body represent the poses of coordinate frames of interest on a body with respect to the body's reference frame.

The simplest examples we looked at in Section II are serial-link manipulators with multiple end effectors or multiple manipulators attached to a base. *Mechanism_Model's* capability for modeling branching kinematic chains makes it easy to model these types of systems. Wheeled, limbed, and hybrid locomotors, discussed in sections II.C., II.D. and II.E of this paper, are modeled by inserting a virtual 6 DOF spatial joint between the ground body and the locomotor chassis. The suspensions, legs or limbs are then modeled as branches from the chassis body. For these examples, spatial constraints between wheel or leg and ground are applied to solve for the chassis pose. Parallel kinematic structures or closed chain mechanisms can be modeled similarly with closure constraints used to specify the attachments of the parallel links to ground or the mechanism closure respectively. The solution approach will use a numerical solver to fully determine the parallel-link manipulator end effector pose or closed chain configuration.

C. Generic Algorithms

With this model framework in place, it is possible to develop generic kinematic and other algorithms that can be re-used for many applications. The most commonly used algorithms in robot control are the forward and inverse kinematics algorithms. In *Mechanism_Model*, these algorithms are designed to be generalized for the tree kinematic structure. The implementation of the forward kinematics allows any frame on the tree to query for its pose with respect to any another frame on the tree for a given set of joint values of the elements in the tree. The generic inverse kinematics algorithm is to be written after we complete implementing a generic constraint solution component of *Mechanism_Model*. This component will allow multiple spatial constraints to be applied at different frames on the tree. It will solve for the corresponding joint values with a constrained optimization algorithm.

Within the same model framework, generic algorithms for wheeled locomotors have also been implemented. We have implemented flat-terrain forward and inverse kinematics algorithms for position and velocity. The position and velocity inverse kinematics algorithms compute wheel steer and drive distances or drive velocities corresponding to a specified body transformation or body velocity respectively. The position and velocity forward kinematics algorithms compute the reverse; they compute the body pose or velocity corresponding to the set of wheel steer and drive distances or drive velocities respectively. These algorithms are parameterized for the number and locations of wheels and for rover type. Supported rover types include fully-

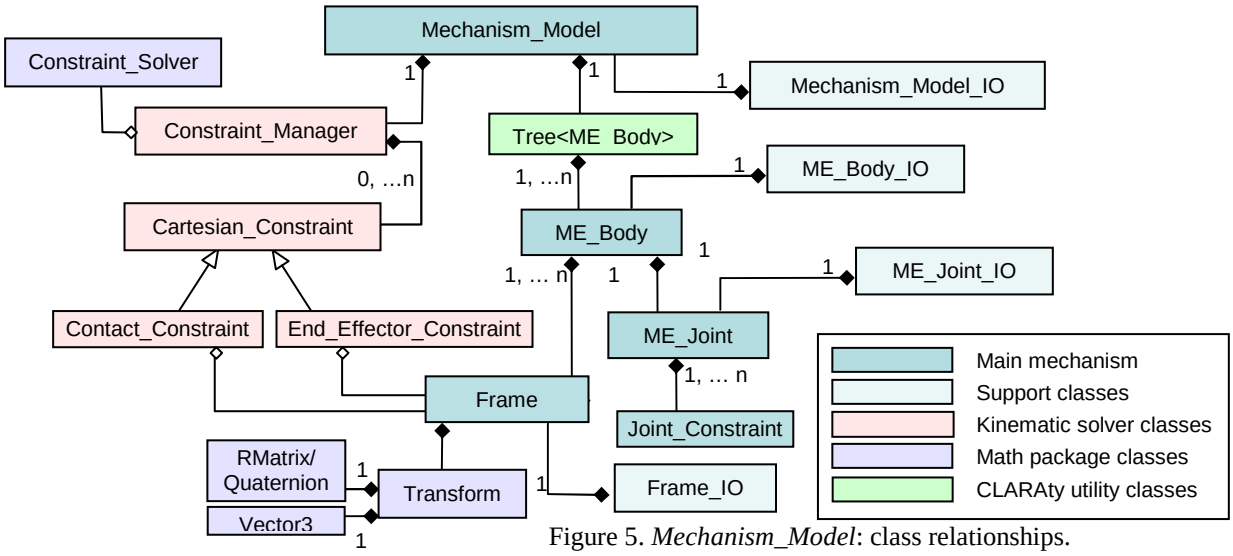


Figure 5. *Mechanism_Model*: class relationships.

steered, partially-steered and skid-steered rovers. These vehicle kinematics algorithms are being extended to handle non-flat terrain and six DOF rover kinematics.

D. Software Implementation

Mechanism_Model is built upon a hierarchy of templatized software utility objects that are written in C++. These objects include a Standard Template Library (STL)-like tree, one and two dimensional array, three dimensional vector, matrix, and rotation matrix and quaternion based homogenous transform classes. Pre-order, post-order, sibling and chain iterators are implemented in the Tree class to facilitate tree traversal for kinematics algorithms.

The *Mechanism_Model* package is composed of Mechanism_Model, ME_Body, ME_Joint and Frame classes. A corresponding set of input-output (I/O) classes, Mechanism_Model_IO, ME_Body_IO, ME_Joint_IO and Frame_IO, are used to read and write model data from and to XML [19] model files. The constraint optimization component of *Mechanism_Model* includes Constraint_Manager, Constraint_Solver, Cartesian_Constraint, Contact_Constraint, and End_Effector_Constraint classes. The constraint solution component of *Mechanism_Model* has been designed and will be implemented in the next stage of development. The relationship between these classes is shown on the UML diagram on Figure 5.

IV. Application

We developed several example applications to validate this approach and illustrate the use of *Mechanism_Model*. A simple 2 DOF planar manipulator is described here to show how a model is created and used with *Mechanism_Model*. There are two ways to create this model. The model can be created using manually entered lines of code. Alternatively, a model can be read in from an XML model file. In our example, the two links of the arm are each 1.0 meter long. The two joints of the arm rotate about the Z-axis. In its nominal configuration with its joints at 0 radians, the arm is aligned along the X-axis.

A new class, N_2DOF_Planar_Arm was derived from *Mechanism_Model* to demonstrate the implementation of a customized inverse kinematics algorithm. The code below shows how the two-link planar manipulator is created in N_2DOF_Planar_Arm.

```
N_2DOF_Planar_Arm arm;

// Create the first link
ME_Body link1("link1", arm);
link1.create_frame("ref1");
link1.get_joint().set_type("revolute");
link1.get_joint().set_joint_axes(
  Vector3<double>(0.0, 0.0, 1.0));

// Create the second link and attach to link1
ME_Body link2("link2", arm, "link1");
Frame & link2_ref_frame =
  link2.create_frame("ref2");

link2_ref_frame.set(
    Transform(Vector3<double>(1.0,0.0,0.0)));
link2.get_joint().set_type("revolute");
link2.get_joint().set_joint_axes(
  Vector3<double>(0.0, 0.0, 1.0));

// Create a local frame at the tip
Frame & link2_tip_frame =
  link2.create_frame("tip");
link2_tip_frame.set(
    Transform(Vector3<double>(1.0,0.0,0.0)));

arm.initialize();
```

A model can, alternatively, be created by reading in the XML model file shown on Figure 6 as follows:

```
N_2DOF_Planar_Arm arm("2dof_planar_arm.xml");
```

To perform the forward kinematics for a given set of joint angles, we first set the arm joint angles, then query for the tip position:

```
arm.get_joint(0).set_value(M_PI_2);
arm.get_joint(1).set_value(0);

Transform tip_position =
arm.get_body("link2").get_frame(
     "tip").get_absolute_transform();
```

```
<Mechanism_Model name = "2dof_planar_arm" version = "1.0">
<!-- This body is first link -->
  <ME_Body name= "link1" >
     <ME_Joint name = "joint1" type = "revolute"
               x_axis = "0"y_axis = "0" z_axis = "1">
     </ME_Joint>
      <Frame name="ref1" type="reference">
        <Transform>
         <Position x="0" y="0" z="0" />
         <Quaternion qi="0" qj="0" qk="0"qs="1" />
        </Transform>
     </Frame>
  </ME_Body>

<!-- This body is second link -->
  <ME_Body  name= "link2" parent = "link1" >
    <ME_Joint name = "joint2" type = "revolute"
              x_axis = "0"y_axis = "0" z_axis = "1">
   </ME_Joint>
      <Frame name="ref2" type="reference">
       <Transform>
          <Position x="1"y="0" z="0" />
         <Quaternion qi="0" qj="0" qk="0"qs="1" />
        </Transform>
     </Frame>
      <Frame name="tip" type="local">
        <Transform>
         <Position x="1" y="0" z="0" />
         <Quaternion qi="0.0" qj="0" qk="0"qs="1.0" />
        </Transform>
     </Frame>
  </ME_Body>
</Mechanism_Model>
```

Figure 6. 2dof_planar_arm.xml XML input file for a 2 dof planar manipulator.

To perform the inverse kinematics is just as easy. We first specify a desired tip location then query for the corresponding arm configuration.

```
Vector<Vector<double>> joint_values;

Vector2<double> tip(1.0,1.0);

arm.inverse_kinematics(tip,joint_values);
```

The two sets of possible results are contained in the 2-D vector of 2-D vectors.

V. Conclusion

The *Mechanism_Model* framework has been successfully used to model the following manipulator arms and rover vehicles: 1) Two-dof planar manipulator, 2) Three-dof modeled with DH parameters, 4) Two-dof Rocky8 mast camera arm, 5) Four-dof Rocky8 mast camera arm, 6) Five-dof Rocky8 instrument arm, 7) FIDO rover with rocker-bogey suspension kinematics, 8) Rocky8 rover with rocker-bogey suspension kinematics, and 9) Rocky8 rover kinematics with a 5-dof manipulator arm. We are continuing its development in a couple of directions. *Mechanism_Model* is about to be interfaced to a new generalized trajectory generation and path planning module and a new generalized control module.

Also in the next stage of development, the constrained solution components of *Mechanism_Model* will be implemented. An iterative constrained optimization algorithm will be used to solve for multiple constraints acting simultaneously on frames in the model. This generalized approach will be useful for many applications including solving the inverse kinematics of manipulator arms and allow positioning an end effector at a desired pose, determining legs placement configuration for a walking machine and determining suspension compliance in a rover with wheels.

With the integration of these components in CLARAty, we will be able to deploy a complete control software package for any of the platforms listed in Section II by merely reading in the system XML configuration files. We expect to see significant reduction in the time it requires to develop control software for robotic platforms with this capability. By end of May 2007, a pilot version of the software described in this paper will be released with the CLARAty public release.

Acknowledgment

This work was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[1] Bickler, D. “A New Family of JPL Planetary Surface Vehicles”, In Missions, Technologies, and Design of Planetary Mobile Vehicle, pages 301-306, Toulouse, France, September 28-30,

[2] Brooks, A., Kaupp, T., Makarenko, A., Williams, S. & Oreback, A. (2005). Towards Component-Based Robotics, Principles and Practices of Software Development in Robotics (SDIR2005), ICRA2005 Workshop, Barcelona, Spain, April 2005.

[3] CLARAty (2007), http://claraty.jpl.nasa.gov

[4] DARTS Lab (2007) http://dshell.jpl.nasa.gov/

[5] A. Diaz-Calderon, I.A. Nesnas, W.S. Kim, and H. Nayar, "Towards a Unified Representation of Mechanisms for Robotic Control Software," International Journal of Advanced Robotic Systems, Vol. 3, No. 1, pp. 061-066, 2006

[6] Jain, A. (1991). Unified Formulation of Dynamics for Serial Rigid Multibody Systems, Journal of Guidance, Control and Dynamics, vol. 14, pp. 531-542.

[7] Kapoor, C. & Tesar, D. (1998). A reusable operational software architecture for advanced robotics, Proceedings of the Twelfth CSIM-IFToMM Symposium on theory and Practice of Robots and Manipulators, Paris, France, July 1998.

[8] B. Kennedy, H. Agazarian, Y. Cheng, M. Garrett, G. Hickey, T. Huntsberger, L. Magnone, C. Mahoney, A. Meyer, J. Knight, Autonomous Robots, Vol. 11, No. 11, 2001, pp. 201-205.

[9] Nesnas, I.A., chapter in Software Engineering for Experimental Robotics, Springer Tracts on Advanced Robotics, edited by Davide Brugali, 2006

[10] Nesnas, I.A., Wright, A., Bajracharya, M., Simmons, R. & Estlin, T. (2003). CLARAty and Challenges of Developing Interoperable Robotic Software, International Conference on Intelligent Robots and Systems (IROS), Nevada, October 2003.

[11] Nucleus (2005). http://www.energid.com/site/nucleus.htm

[12] ORCA (2005). http://orca-robotics.sourceforge.net/index.html

[13] OROCOS (2007). http://www.orocos.org/

[14] OSCAR (2005). http://www.robotics.utexas.edu/rrg/research/oscarv.2/

[15] RoboML (2005). http://www.roboml.org/

[16] Rodriguez, G., Kreutz-Delgado, K. & Jain, A. (1991). A Spatial Operator Algebra for Manipulator Modeling and Control, International Journal of Robotics Research, vol. 10, pp. 371-381.

[17] Volpe, R., Nesnas, I.A.D., Estlin, T., Mutz, D., Petras, R. & Das, H. (2001). The CLARAty Architecture for Robotic Autonomy, Proceedings of the 2001 IEEE Aerospace Conference, Big Sky Montana, March 2001.

[18] Wilcox, B., ATHLETE: A Landing, Mobility, and Manipulation System for the Moon, to be presented at ICRA '07 Space Robotics Workshop.

[19] The World Wide Web Consortium (2005). Extensible Markup Language (XML), htttp://www.w3.org/XML