Using Symbolic Technology to Derive Inverse Kinematic Solutions for Actuator Control Development

P. Goossens*. T. Richard**

**Maplesoft, Waterloo, ON N2V 1K8*
Canada (Tel: 519-747-2373; e-mail: pgoossens@maplesoft.com).
***Maplesoft, Auf der Hüls 198 52068 Aachen, DE (e-mail:*
trichard@maplesoft.com)

Abstract: In multibody mechanics, the motion analysis for a platform (the kinematics problem) can be classified into two cases: the forward kinematics and the inverse kinematics problems. For the forward kinematics problem, the trajectory of a point on a mechanism (for example, the end effector of a robot arm or the center of a platform support by a parallel link manipulator) is computed as a function of the joint motions. In the inverse kinematics case, the problem is reversed: the goal is to compute the joint motions necessary to achieve a prescribed end effector trajectory.

In general, given the mechanism geometry, it is quite straightforward to solve the forward kinematics problem both numerically and symbolically. In contrast, solving for the inverse kinematic problem typically involves solving a nonlinear system or equation with trigonometric functions. Issues such as singularity, multiple solutions (as in the case of "elbow up" and "elbow down" configurations for a robot arm), and no solution (as in the case in which the specified trajectory goes beyond the workspace of the mechanism) can often come up, further complicating the solution process. The complexity in the inverse kinematics problem is compounded even more for parallel link manipulators.

Because of the complexity involved, the inverse kinematics problem is often solved numerically through iterations, and is computationally expensive. With a numeric approach, however, information about the motion of the mechanism is often lost. In this paper, we will describe how to obtain a symbolic solution to the inverse kinematics problem for two real problems using tools available in MapleSim™. The first, a 2 degrees-of-freedom (DOF) tracking radar gimbal is used to show the principal steps in a relatively simple mechanism. These principles are then demonstrated with a much more complex mechanism: a Stewart-Gough hydraulic platform.

Furthermore, we will show how having access to the symbolic Jacobian of the constraint equations allows us to inspect and exploit the underlying matrix structure, which leads to a simplified solution process for obtaining the symbolic solution.

An advantage of having a symbolic solution to the inverse kinematics problem is the possibility of code-generating the symbolic solution so that it can be embedded in real-time hardware-in-the-loop (HIL) applications. This approach can be contrasted with a purely numerical approach where the iterative solution process makes it difficult to use in real-time applications.

Keywords: inverse kinematic problems, kinematics, modeling, simulation, mechanical engineering, mechanical systems, computer-aided engineering, computer-aided system design

1. INTRODUCTION

In multibody mechanics, the motion analysis for a platform (the kinematics problem) can be classified into two cases: the forward kinematics and the inverse kinematics problems. For the forward kinematics problem, the trajectory of a point on a mechanism (for example, the end effector of a robot arm or the center of a platform support by a parallel link manipulator) is computed as a function of the joint motions. In the inverse kinematics case, the problem is reversed: the goal is to compute the joint motions necessary to achieve a prescribed end effector trajectory.

In general, given the mechanism geometry, it is quite straightforward to solve the forward kinematics problem both numerically and symbolically. In contrast, solving for the inverse kinematic problem typically involves solving a nonlinear system or equation with trigonometric functions. Issues such as singularity, multiple solutions (as in the case of "elbow up" and "elbow down" configurations for a robot arm), and no solution (as in the case in which the specified trajectory goes beyond the workspace of the mechanism) can often come up, further complicating the solution process. The complexity in the inverse kinematics problem is compounded even more for parallel link manipulators.

Because of the complexity involved, the inverse kinematics problem is often solved numerically through iterations, and is computationally expensive. With a numeric approach, however, information about the motion of the mechanism is often lost. In this paper, we will describe how to obtain a symbolic solution to the inverse kinematics problem for two real problems using tools available in MapleSim™. The first, a 2 degrees-of-freedom (DOF) tracking radar gimbal is used to show the principal steps in a relatively simple mechanism. These principles are then demonstrated with a much more complex mechanism: a Stewart-Gough hydraulic platform.

Furthermore, we will show how having access to the symbolic Jacobian of the constraint equations allows us to inspect and exploit the underlying matrix structure, which leads to a simplified solution process for obtaining the symbolic solution.

An advantage of having a symbolic solution to the inverse kinematics problem is the possibility of code-generating the symbolic solution so that it can be embedded in real-time hardware-in-the-loop (HIL) applications. This approach can be contrasted with a purely numerical approach where the iterative solution process makes it difficult to use in real-time applications.

2. SYMBOLIC INVERSE KINEMATICS: TRACKING RADAR GIMBAL CONTROL

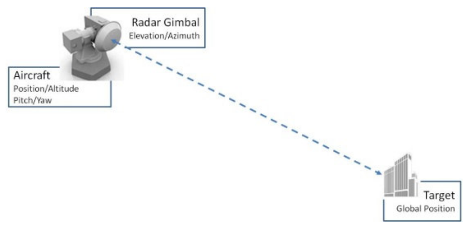


Fig. 1. *Problem: How to define required elevation/azimuth angles of gimbal to maintain contact with target while aircraft is moving?*

A tracking radar on an aircraft uses a 2 DOF gimbal mechanism (elevation and azimuth) to position the radar dish to point at a selected target, typically located by a global positioning system that provides the latitude, longitude and altitude (X, Y, Z) of the target. Since the aircraft can approach the target from any position and altitude, and with any orientation, determining the required azimuth and elevation angles is a major challenge for the designers of the servo drives for the gimbal.

A very innovative solution to this problem – and an excellent demonstration of the power of symbolic technology – is the use of Inverse Kinematics to provide an exact solution for the azimuth and elevation angles, given the topology of the mechanism and defining the line of sight between the radar dish and the target as a constraint.

In general terms, Inverse Kinematics (IK) is a robotics technique that is used to determine the required joint angles to cause the end-effector to follow a predetermined path. In this particular case, the aircraft and radar gimbal can be considered as a robotic mechanism, with the line of sight defining the path of the end-effector (the radar dish).

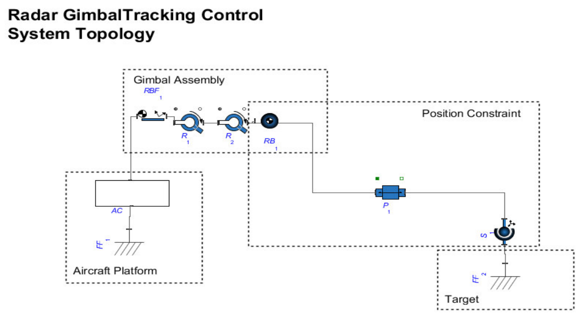


Fig. 2. *Solution: Inverse Kinematics*

In MapleSim, the "mechanism" can be described in three parts: the aircraft, modeled as a 6-DOF platform; the gimbal, modeled as a 2-DOF platform; and the target, defined as a point in space as absolute X, Y, Z coordinates. The line of sight is defined as a position constraint, connected between the centre of the radar gimbal and a spherical joint at the target, with a prismatic joint. This mechanically connects the gimbal to the target, making the gimbal angles mathematically deterministic.

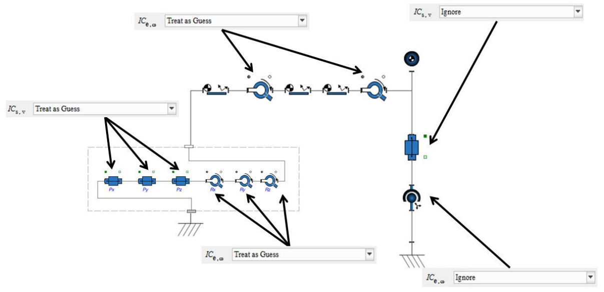
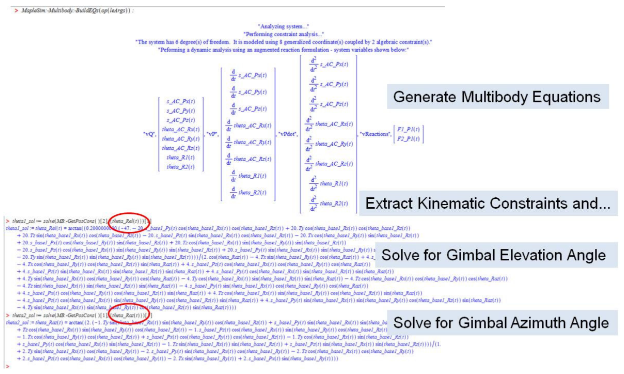


Fig. 3. *MapleSim model preparation*

In a little more detail: the aircraft model uses MapleSim multibody components - three prismatic joints to define the X, Y and Z translation motion, and three revolute joints to define the rotational motion (pitch, roll, yaw). Similarly, the gimbal model uses two revolute joints for the azimuth and elevation rotations.

Any of these joints can be used to drive the aircraft motion in flight as well as the direction of the radar dish. Each joint is uniquely named which helps to identify the required variables when the equations of motion are generated. Furthermore, to ensure the required variables appear in the equations, the constraints need to be explicitly included in the model. This can be achieved by setting the appropriate options for each of the joints.

Once the mechanism model has been created in MapleSim, the model can be accessed in Maple™ and, using its multibody analysis tools, the equations of motion for the mechanism can be automatically extracted.



```
> MapleSim:-Multibody:-BuildEQs(op(leArgs)) :
                              "Analyzing system..."
                          "Performing constraint analysis..."
   "The system has 6 degree(s) of freedom.  It is modeled using 8 generalized coordinate(s) coupled by 2 algebraic constraint(s)."
       "Peforming a dynamic analysis using an augmented reaction formulation - system variables shown below:"
```

$$"vQ", \begin{bmatrix} s\_AC\_Px(t) \\ s\_AC\_Py(t) \\ s\_AC\_Pz(t) \\ theta\_AC\_Rx(t) \\ theta\_AC\_Ry(t) \\ theta\_AC\_Rz(t) \\ theta\_R1(t) \\ theta\_R2(t) \end{bmatrix}, "vP", \begin{bmatrix} \frac{d}{dt} s\_AC\_Px(t) \\ \frac{d}{dt} s\_AC\_Py(t) \\ \frac{d}{dt} s\_AC\_Pz(t) \\ \frac{d}{dt} theta\_AC\_Rx(t) \\ \frac{d}{dt} theta\_AC\_Ry(t) \\ \frac{d}{dt} theta\_AC\_Rz(t) \\ \frac{d}{dt} theta\_R1(t) \\ \frac{d}{dt} theta\_R2(t) \end{bmatrix}, "vPdot", \begin{bmatrix} \frac{d^2}{dt^2} s\_AC\_Px(t) \\ \frac{d^2}{dt^2} s\_AC\_Py(t) \\ \frac{d^2}{dt^2} s\_AC\_Pz(t) \\ \frac{d^2}{dt^2} theta\_AC\_Rx(t) \\ \frac{d^2}{dt^2} theta\_AC\_Ry(t) \\ \frac{d^2}{dt^2} theta\_AC\_Rz(t) \\ \frac{d^2}{dt^2} theta\_R1(t) \\ \frac{d^2}{dt^2} theta\_R2(t) \end{bmatrix}, "vReactions", \begin{bmatrix} F1\_P1(t) \\ F2\_P1(t) \end{bmatrix}$$

```
> theta1_sol := solve(MB:-GetPosCons( )[2], {theta_Rel(t)})[1]
theta1_sol := theta_Rel(t) = arctan((0.2000000000 (-47. - 20. s_base1_Py(t) cos(theta_base1_Rx(t)) cos(theta_base1_Rz(t)) + 20. Ty cos(theta_base1_Rx(t)) cos(theta_base1_Rz(t))
    + 20. Tz sin(theta_base1_Rx(t)) cos(theta_base1_Rz(t)) - 20. s_base1_Pz(t) sin(theta_base1_Rx(t)) cos(theta_base1_Rz(t)) - 20. Tx cos(theta_base1_Ry(t)) sin(theta_base1_Rz(t))
    + 20. s_base1_Px(t) cos(theta_base1_Ry(t)) sin(theta_base1_Rz(t)) + 20. Tz cos(theta_base1_Rx(t)) sin(theta_base1_Ry(t)) sin(theta_base1_Rz(t))
    - 20. s_base1_Pz(t) cos(theta_base1_Rx(t)) sin(theta_base1_Ry(t)) sin(theta_base1_Rz(t)) + 20. s_base1_Py(t) sin(theta_base1_Rx(t)) sin(theta_base1_Ry(t)) s
    - 20. Ty sin(theta_base1_Rx(t)) sin(theta_base1_Ry(t)) sin(theta_base1_Rz(t))))/(2. cos(theta_Raz(t)) - 4. Tx sin(theta_base1_Ry(t)) cos(theta_Raz(t)) + 4. s_
    - 4. Tx cos(theta_base1_Ry(t)) cos(theta_base1_Rz(t)) sin(theta_Raz(t)) + 4. s_base1_Pz(t) cos(theta_base1_Rx(t)) cos(theta_base1_Ry(t)) cos(theta_Raz(t))
    + 4. s_base1_Pz(t) sin(theta_base1_Rx(t)) sin(theta_base1_Rz(t)) sin(theta_Raz(t)) + 4. s_base1_Py(t) cos(theta_base1_Rx(t)) sin(theta_base1_Rz(t)) sin(theta_Raz(t))
    + 4. Ty sin(theta_base1_Rx(t)) cos(theta_base1_Ry(t)) cos(theta_Raz(t)) - 4. Ty cos(theta_base1_Rx(t)) sin(theta_base1_Rz(t)) sin(theta_Raz(t)) - 4. Tz cos(theta_base1_Rx(t)) cos(theta_base1_Ry(t)) cos(theta_Raz(t))
    - 4. Tz sin(theta_base1_Rx(t)) sin(theta_base1_Rz(t)) sin(theta_Raz(t)) - 4. s_base1_Py(t) sin(theta_base1_Rx(t)) cos(theta_base1_Ry(t)) cos(theta_Raz(t))
    + 4. s_base1_Px(t) cos(theta_base1_Ry(t)) cos(theta_base1_Rz(t)) sin(theta_Raz(t)) + 4. Tz cos(theta_base1_Rx(t)) sin(theta_base1_Ry(t)) cos(theta_base1_Rz(t)) sin(theta_Raz(t))
    - 4. s_base1_Pz(t) cos(theta_base1_Rx(t)) sin(theta_base1_Ry(t)) cos(theta_base1_Rz(t)) sin(theta_Raz(t)) + 4. s_base1_Py(t) sin(theta_base1_Rx(t)) sin(theta_base1_Ry(t)) cos(theta_base1_Rz(t)) sin(theta_Raz(t))
    - 4. Ty sin(theta_base1_Rx(t)) sin(theta_base1_Ry(t)) cos(theta_base1_Rz(t)) sin(theta_Raz(t))))
> theta2_sol := solve(MB:-GetPosCons( )[1], {theta_Raz(t)})[1]
theta2_sol := theta_Raz(t) = arctan((2. (-1. Ty sin(theta_base1_Rx(t)) sin(theta_base1_Ry(t)) cos(theta_base1_Rz(t)) + s_base1_Py(t) sin(theta_base1_Rx(t)) sin(theta_ba
    + Tz cos(theta_base1_Rx(t)) sin(theta_base1_Ry(t)) cos(theta_base1_Rz(t)) - 1. s_base1_Pz(t) cos(theta_base1_Rx(t)) sin(theta_base1_Ry(t)) cos(theta_base1_Rz(t)
    - 1. Tx cos(theta_base1_Ry(t)) cos(theta_base1_Rz(t)) + s_base1_Px(t) cos(theta_base1_Ry(t)) cos(theta_base1_Rz(t)) - 1. Ty cos(theta_base1_Rx(t)) sin(theta_base1_Rz(t))
    + s_base1_Py(t) cos(theta_base1_Rx(t)) sin(theta_base1_Rz(t)) - 1. Tz sin(theta_base1_Rx(t)) sin(theta_base1_Rz(t)) + s_base1_Pz(t) sin(theta_base1_Rx(t)) sin(theta_base1_Rz(t))))/(1.
    + 2. Ty sin(theta_base1_Rx(t)) cos(theta_base1_Ry(t)) - 2. s_base1_Py(t) sin(theta_base1_Rx(t)) cos(theta_base1_Ry(t)) - 2. Tz cos(theta_base1_Rx(t)) cos(theta_base1_Ry(t))
    + 2. s_base1_Pz(t) cos(theta_base1_Rx(t)) cos(theta_base1_Ry(t)) - 2. Tx sin(theta_base1_Ry(t)) + 2. s_base1_Px(t) sin(theta_base1_Ry(t))))
>
```

Generate Multibody Equations
Extract Kinematic Constraints and...
Solve for Gimbal Elevation Angle
Solve for Gimbal Azimuth Angle

Fig. 4. *Inverse Kinematic solution in Maple*

Figure 4 shows the resulting equations of motion in Maple. The Multibody:-BuildEQs() command extracts the relationships defined by the mechanism topology given in the MapleSim model and combines them appropriately to generate the dynamic and kinematic equations of motion. This includes the mathematical expressions for the kinematic constraints in terms of the rotational and translational joints.

The two constraint equations take the form: $f_i(\theta_{az}, \theta_{el}, \mathbf{r}, \xi, \eta, \zeta) = 0, \ \ i = 1,2$

, where $\mathbf{r}$ is the vector (aircraft) body position and ξ, η, ζ are the body orientation angles. Also θ_{az}, θ_{el} are the azimuth and elevation angles of the gimbal, respectively.

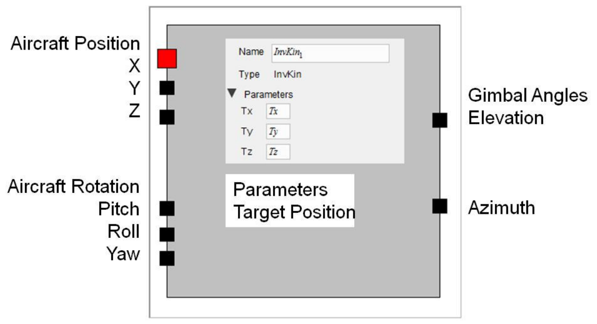
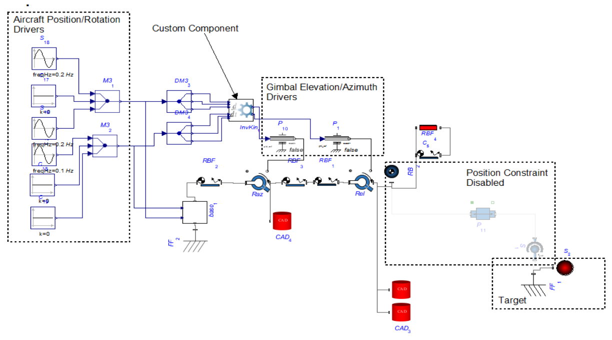


Fig. 5. *MapleSim custom component*

To implement these symbolic constraint expressions in the MapleSim model, the equations can be used to create a custom component, taking the aircraft position and rotation as input signals and returning the required Elevation and Azimuth angles. The absolute coordinates of the target position are entered as parameters. The Maple algebraic solver will solve for the Elevation and Azimuth angles automatically when the simulation is executed.



Aircraft Position/Rotation Drivers
Custom Component
Gimbal Elevation/Azimuth Drivers
Position Constraint Disabled
Target

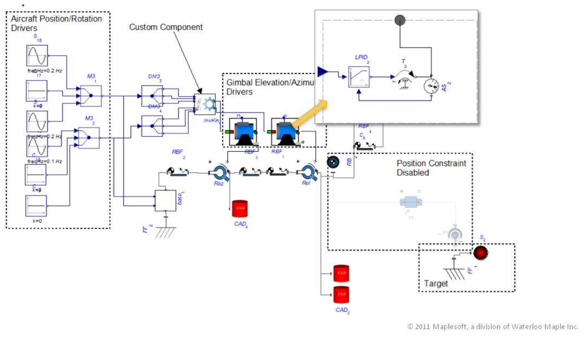
Fig. 6. *Implementation of Inverse Kinematic solution in MapleSim*

The resulting custom component is used to modify the earlier model (see Figure 2) by connecting the angle outputs to the elevation and azimuth joints in the gimbal model by way of 1D rotational motion drivers. To drive the model, a file of aircraft flight data can be used to provide the required translational and rotational motions of the aircraft. However, for the purpose of this initial test, a simple set of sine-wave signals are used. These signals are sent to the aircraft platform model as well as the inputs to the custom component.

The line-of-sight position constraint can now be disabled since the results from the custom component will now drive the joint angles.

The best way of testing if the generated angles are correct is to use MapleSim's 3-D visualization capability to produce an animation of the resulting simulation. A video about this model can be found on the Maplesoft website. You will see from this video that the tracking "beam" – a perpendicular line that extends from the centre of the radar dish – always goes through the target as the "aircraft" moves around.

After testing that the angles from the IK solution are correct, they can then be used as set-point angles for the servo motors on the gimbal.



Aircraft Position/Rotation Drivers
Custom Component
Gimbal Elevation/Azimu Drivers
Position Constraint Disabled
Target
© 2011 Maplesoft, a division of Waterloo Maple Inc.

Fig. 7. *Implementation of servo drives*

In this example, the motors are modelled as torque drivers with the required torque being calculated by a PID controller. These replace the motion drivers in the model for the elevation and azimuth.

After some tuning of the controllers, the resulting responses on Figure 8 show the actual joint angles stabilize very quickly and track the IK angles very closely thereafter.

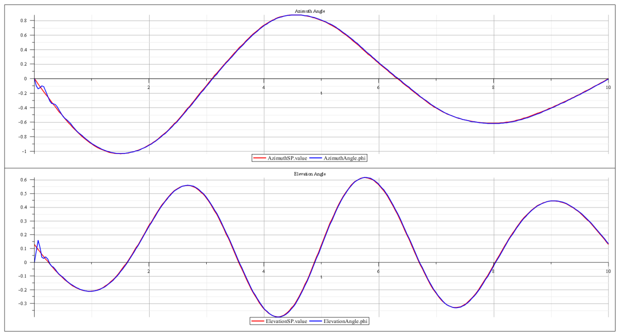


Fig. 8. *Time response of servo controlled joint angles*

3. STEWART-GOUGH PLATFORM

The second example we present in this paper is a parallel robot that once was considered one of the most difficult problems of robot kinematics, the Stewart-Gough platform. The Stewart-Gough manipulator is a six-DOF parallel linkage mechanism. There are several variations in the leg configurations of a Stewart-Gough platform. In this paper, we will be looking at a Stewart-Gough (see Figure 9) whose motion is provided through six prismatic joints; each is anchored with a universal joint (two cascaded revolute joints) on one end and a spherical joint on the other end as shown in Figure 10). Each of the platform legs is defined with five parameters, specifying the x-y offset of the two anchor points (Xg, Yg, Xp, and Yp) and the base anchor rotation angle offset (Ang). The inverse kinematics problem in this case is to find the length of each of the six prismatic joints that will result in a desired motion trajectory for the supported platform.

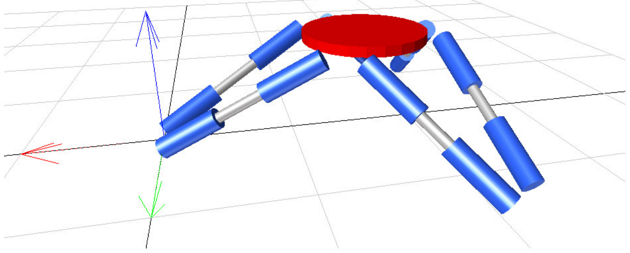


Fig. 9. *Screenshot of Stewart-Gough platform in MapleSim*

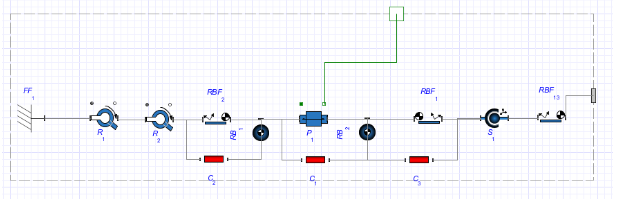
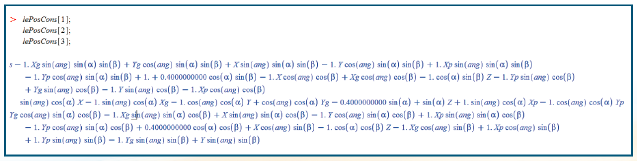


Fig. 10. *Platform leg structure*

In a similar manner, we can use a specialized set of Maple commands to retrieve the symbolic system equations for the Stewart-Gough platform and then obtain a set of 18 constraint equations for the six prismatic joint lengths, plus 12 revolute joint angles for the orientation of the legs. In addition to being a nontrivial problem to solve, the sheer size of this inverse problem implies that even with numeric iterations, the calculations involved are still quite intense. Further, the numeric solution would not provide any insights into the structure of the problem that might be exploited for design or performance considerations.

As it turns out, for this particular leg configuration, a structural property can be exploited to simplify the solution of the inverse kinematic problem so that a closed-form symbolic solution can be obtained. The 18 position constraint equations come from the spherical joints that are used to attach each leg to the platform, which result from enforcing the x, y, z position of each leg's end to be coincident with a given point on the platform. Such structural properties can be observed from the definition of the state vector and the 18x24 position constraint Jacobian matrix (for 18 constraints and 24 generalized coordinates) that can be easily inspected using the matrix browser in the Maple template. The constraint equations vector are bundled into six groups corresponding to the variables associated with each of the six legs in the system that can be solved independently. Doing so, we can transform the original problem of solving 18 nonlinear constraint equations into a problem of solving six sets of three nonlinear constraint equations independently. The overall inverse kinematic problem can be further stripped down to only solving a subset of three position constraint equations by taking advantage of the symmetry and parametric nature of the leg component. The reduced set of position constraint equations are shown in Figure 11.



```
> lePosCons[1];
  lePosCons[2];
  lePosCons[3];
```

$$s - 1.\,Xg \sin(ang) \sin(\alpha) \sin(\beta) + Yg \cos(ang) \sin(\alpha) \sin(\beta) + X \sin(ang) \sin(\alpha) \sin(\beta) - 1.\,Y \cos(ang) \sin(\alpha) \sin(\beta) + 1.\,Xp \sin(ang) \sin(\alpha) \sin(\beta) - 1.\,Yp \cos(ang) \sin(\alpha) \sin(\beta) + 1. + 0.4000000000 \cos(\alpha) \sin(\beta) - 1.\,X \cos(ang) \cos(\beta) + Xg \cos(ang) \cos(\beta) - 1.\cos(\alpha) \sin(\beta)\, Z - 1.\,Yp \sin(ang) \cos(\beta) + Yg \sin(ang) \cos(\beta) - 1.\,Y \sin(ang) \cos(\beta) - 1.\,Xp \cos(ang) \cos(\beta)$$

$$\sin(ang) \cos(\alpha)\, X - 1.\sin(ang) \cos(\alpha)\, Xg - 1.\cos(ang) \cos(\alpha)\, Y + \cos(ang) \cos(\alpha)\, Yg - 0.4000000000 \sin(\alpha) + \sin(\alpha)\, Z + 1.\sin(ang) \cos(\alpha)\, Xp - 1.\cos(ang) \cos(\alpha)\, Yp$$

$$Yg \cos(ang) \sin(\alpha) \cos(\beta) - 1.\,Xg \sin(ang) \sin(\alpha) \cos(\beta) + X \sin(ang) \sin(\alpha) \cos(\beta) - 1.\,Y \cos(ang) \sin(\alpha) \cos(\beta) + 1.\,Xp \sin(ang) \sin(\alpha) \cos(\beta) - 1.\,Yp \cos(ang) \sin(\alpha) \cos(\beta) + 0.4000000000 \cos(\alpha) \cos(\beta) + X \cos(ang) \sin(\beta) - 1.\cos(\alpha) \cos(\beta)\, Z - 1.\,Xg \cos(ang) \sin(\beta) + 1.\,Xp \cos(ang) \sin(\beta) + 1.\,Yp \sin(ang) \sin(\beta) - 1.\,Yg \sin(ang) \sin(\beta) + Y \sin(ang) \sin(\beta)$$

Fig. 11. *A subset of position constraints*

With this simplified problem, we can now readily implement these equations as a custom component in MapleSim to obtain an inverse kinematics simulation. The result of the simulation is shown in Figure 12.

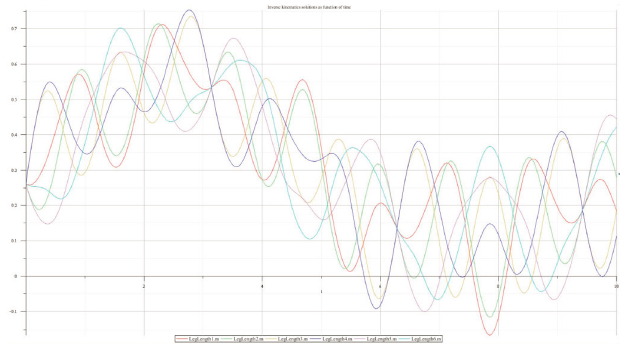


Fig. 11. *Inverse Kinematics solution*

4. CONCLUSION

Inverse kinematics is a highly advanced approach to solving motion planning problems. While there are many benefits to using an inverse kinematic approach, the difficulties in solving these problems manually mean that often non-robust and iterative techniques are used instead. In this paper, we have shown how inverse kinematics problems can be solved by taking advantage of MapleSim's ability to access the underlying symbolic system and exploit the symbolic structure of the Jacobian matrix. It should be noted that the same problem can also be solved numerically within MapleSim by applying enforced motion to the system by using a Prescribed Translation motion driver component. However, such a purely numerical solution does not have the same flexibility as the symbolic solution presented in this paper. Using the symbolic solution obtained here and MapleSim's code generation feature, efficient simulation code can be obtained and embedded into other platforms, unlocking the possibility for real-time applications.

5. REFERENCES

Solving inverse kinematic problems: tracking-radar motion control. (Online), November 15, 2011. http://www.maplesoft.com/products/maplesim/demo/inverse_kinematic_gimbal.aspx.