An Example of Inverse and Forward Kinematics Using an ACS SPiiPlus Controller:
The 6 DOF Stewart Platform Hexapod

1. Introduction

Many robotic and mechanical systems require the calculation of kinematic equations to express the relationship between variables that are to be controlled (motor/actuator position obtained via feedback sensors and manipulated by motors/actuators) and variables that are to be commanded (i.e. position of a tool tip or objective). Two types of equations are necessary for employing such systems – Inverse and Forward Kinematic Equations or Kinematics. Inverse kinematics provides solutions to the following situation: if one knows the desired values for commanded variables, what should the corresponding controlled variables be? Conversely, forward kinematics determine what the commanded variables should be if the controlled variables are known. This document will familiarize the reader with kinematic equations and general implementation in the SPiiPlus controller series, as
well as demonstrate the ability of ACS Motion Control’s SPiiPlus controller software
to implement kinematic equations to control a 6 DOF hexapod robot.

2. A Simple Kinematics Example – The Rotary Tool Tip

To get a good idea of what kinematic equations are, a simple example is
presented here, which will give some insight to the more complicated hexapod
system. Suppose a system consists of an XY motor stage, and on the XY stage is a
rotary motor that controls the angle of a tool (T) with length (from the rotation pivot)
L. This is depicted in Figure 1 below.

![Figure 1 – Geometry of Tool Tip System](image)

The user will want to control the angle T, and the location of the tool tip in the
XY plane \((x_T, y_T)\). These are the commanded variables. The controlled variables are
the angle T, and the motor positions \(x_M\) and \(y_M\). The inverse kinematic equation for
this system will tell us: what should the controlled variables be for a desired set of
commanded variables. Often, inverse kinematic equations are less difficult to obtain
than their forward counterparts, and for many systems they are discussed and solved
first. Then, the forward equations are solved by inverting the inverse equations; either
directly or via an iterative step approach (the Hexapod uses the latter). Here both
forward and inverse equations are directly obtainable, and the inverse equations are
shown in Equation 1.
\[ I_1(x_T, y_T, T) = (x_M, y_M, T) \]
\[ \Rightarrow x_M = x_T - L \cos(T) \]
\[ \Rightarrow y_M = y_T - L \sin(T) \]
\[ \Rightarrow T = T \tag{1} \]

From equation 1, it is obvious that there are 3 equations comprising the inverse kinematics, although the last equation \((T = T)\) is redundant. In this system, determining the forward kinematic equations is straightforward and can be done by inspection of the system, or by inverting the equations of (1). The solution to the forward kinematic equations is shown in equation 2.

\[ F_1(x_M, y_M, T) = (x_T, y_T, T) \]
\[ \Rightarrow x_T = x_M + L \cos(T) \]
\[ \Rightarrow y_T = y_M + L \sin(T) \]
\[ \Rightarrow T = T \tag{2} \]

For this system, both the forward and inverse kinematic equations were easily obtained in closed-form solutions. For many systems including the 6 DOF Hexapod, this is not necessarily the case.

3. APOS, RPOS, and FPOS – The Story Behind SPiiPlus Position Servo Variables

The SPiiPlus controller employs a powerful profile generator that constructs 3rd order motion profiles in real time, allowing on the fly profile changing with guaranteed smooth motion. This profile generator sends a commanded position to the servo loop every controller cycle, and then the servo interpolates at an even finer (higher frequency) servo cycle, using feedback from an encoder source. So, it would seem that only 2 position variables would be necessary in this process – the desired position and feedback position. This is true for many controllers, but the SPiiPlus controller actually allows for a much more powerful and advanced scheme to generate profiles by using the CONNECT function.

The CONNECT function is what allows such simple implementation of kinematic equations, by allowing the user to define a mathematical relationship between logical and physical axes. The logical axes are the axes that the user wishes to work with, which corresponds to the ‘commanded’ variables described earlier. Conversely, the physical axes are the axes that are directly related to the physical motors or actuators. These axes have been described in this document as the ‘controlled’ variables. For the tool tip system above, the logical axes would be the tool tip location \((x_T, y_T)\) and angle \(T\). The physical axes would be the motor positions \((x_M, y_M)\) and angle \(T\).
So, we have logical and physical axes, or commanded and controlled variables, and the mathematical relationship between these axes is described by the CONNECT function. Knowing this, it becomes clear why the 2 variables – APOS and RPOS – are needed: APOS corresponds to the logical axes, and RPOS corresponds to the physical axes. In many systems however, there is no need for a distinction between the two, and for these systems, RPOS(axis) = APOS(axis), or in other words, the CONNECT function is equal to 1 (RPOS(axis) = 1*APOS(axis)). An example of this would be an XYZ stage with 3 linear motors – both the commanded and controlled variables are the same - linear positions in X, Y, and Z. In systems where controlled and commanded variables are not the same, which is the case for systems using kinematic equations, the CONNECT function is no longer ‘1’. Instead, it is defined by the inverse kinematic equations of the system. Figure 2 shows how the variables APOS, RPOS, and FPOS are related in the SPiiPlus controller.

For systems involving kinematic equations, the APOS variables simply refer to the desired values of the commanded variables, and the RPOS variables refer to the desired values of the controlled variables. FPOS then contains the measured values of controlled variables, and forward kinematic equations are then needed to obtain the corresponding measured values of the commanded variables (this is discussed in part 5 of this document).

To implement the inverse kinematics from the tool tip system, the following CONNECT functions would be issued in the SPiiPlus controller:

- \[ \text{CONNECT RPOS(0)} = \text{APOS(0)} - L^* \cos(\text{APOS(2)}) \]
- \[ \text{CONNECT RPOS(1)} = \text{APOS(1)} - L^* \sin(\text{APOS(2)}) \]
- \[ \text{CONNECT RPOS(2)} = \text{APOS(2)} \]

In the above equations relating to the tool tip system, APOS(0) = x_T, APOS(1) = y_T, APOS(2) = T, RPOS(0) = x_M, RPOS(1) = y_M, and RPOS(2) = T.
4. Inverse Kinematics of the Hexapod

For the Hexapod parallel manipulator, there are 6 individual linear motors controlling the length of 6 legs. Each of these legs is connected to a base structure, and a platform structure as shown in the picture on page 1. The inverse kinematic equations for this system tell the user: Given a desired platform position \((x, y, z, \alpha, \beta, \gamma)\), what should the lengths of the 6 legs be? The geometry of the system is shown below in Figure 3.

![Figure 3 – Geometry of the Hexapod](image)

After bypassing some geometry and algebra, the inverse kinematics equations can be simply written as a vector function \(I_2\), which takes 6 “commanded” variables as inputs, and produces the 6 “controlled” variables as a result:

\[
I_2(x, y, z, \alpha, \beta, \gamma) = (l_1, l_2, l_3, l_4, l_5, l_6)
\] (3)

For the 6 DOF hexapod, this is in fact a closed-form solution. So, to command the hexapod to move to a certain \((x, y, z, \alpha, \beta, \gamma)\) value, one must compute the inverse kinematics to find the corresponding \((l_1, l_2, l_3, l_4, l_5, l_6)\) values, and then command the legs to move to those positions. This seems pretty easy so far, but how should one command the motors to get to positions \((l_1, l_2, l_3, l_4, l_5, l_6)\)? One cannot simply set the commanded (reference) positions to \((l_1, l_2, l_3, l_4, l_5, l_6)\) or else the motor will instantaneously try to jump to this position (actually the SPiiPlus controller won’t allow this). Instead, motion trajectories must be generated for each of the leg motors. To do this, the powerful CONNECT function is used. The user can simply command the “commanded” variables (in this case, \(x, y, z, \alpha, \beta, \gamma\)) and the inverse kinematics calculation and CONNECT function will automatically move the motors...
accordingly. This allows profiles to be generated in the $x$, $y$, $z$, $\alpha$, $\beta$, and $\gamma$ directions using the SPiiPlus MPU internal motion profile generator! In the case of the hexapod, Figure 4 shows the concept for implementing inverse kinematics using ACSPL+.

Figure 4 – Inverse Kinematics using ACSPL+

Note that in the figure above, the ACSPL+ buffer is needed to continuously calculate the inverse kinematic equations, whose arguments (APOS variables) are updated every controller cycle by the MPU. This is because the motion profile generated by the MPU is in terms of our commanded variables ($x$, $y$, $z$, $\alpha$, $\beta$, $\gamma$), and servo needs to be given it’s commands in terms of the controlled variables ($l_1$, $l_2$, $l_3$, $l_4$, $l_5$, $l_6$). The CONNECT function does not need to be executed every cycle, since the controller automatically evaluates its argument every cycle.

The inverse kinematics buffer starts by defining all of the $x$, $y$, and $z$ positions of the base and platform joints. Then, the initial APOS and RPOS variables are set (where the motors are known to be at their starting positions), after which the first calculation of the inverse kinematics is done. Then, the CONNECT function is issued for all axes, and the loop is entered, which continuously calculates the inverse kinematics and corresponding leg lengths that are used by the CONNECT functions.

To recap, a new profile command is generated every controller cycle (default 1ms for SPiiPlus controllers) by the MPU, which needs to be transformed (via CONNECT functions, whose arguments are calculated by inverse kinematic equations) into a command that can be used by servo algorithm. Here, this means desired positions for each of the leg motors.
5. Forward Kinematics of the Hexapod

Using inverse kinematics as described above, it is possible to send commands in terms of commanded variables and let the servo algorithm work in terms of controlled variables. But once the user commands a move, how does he/she know what is actually happening in terms of the commanded variables? Or, if the axis is disabled, how does the user know the state (positions) of the commanded variables? A specific example regarding the hexapod would be the following. A user disables the hexapod at a certain position. After some time, the user wishes to know: what are values of the commanded variables \((x, y, z, \alpha, \beta, \gamma)\)? Have they changed since disabling the motors at the last commanded position? In this situation, it is likely that the legs have moved some amount, even though the user has not commanded any movement.

Solving this problem requires forward kinematic equations. Simply put, the forward kinematics equations tell the user: what are the values of the commanded variables, given the values of the controlled variables. For the hexapod application, forward kinematics tells the user the position of the platform given the lengths of the legs. Just as we did for the inverse kinematic equation, the forward kinematic equation can be written mathematically as a vector function \(F_2\):

\[
F_2(l_1, l_2, l_3, l_4, l_5, l_6) = (x, y, z, \alpha, \beta, \gamma)
\]  

(4)

For the hexapod in this application, there is not a closed form solution to this problem that is practical for real-time execution. Instead, the solution is most readily obtained by performing an iterative process involving estimating the solution, evaluating the estimation, forming a new estimation, evaluating the new estimation, and so on until the estimation is below some error threshold. This is done in an ACSPL+ buffer as follows.

1. Estimate that the actual measured \((x, y, z, \alpha, \beta, \gamma)\) position is the commanded position. Let’s call this \(E\).
2. Use the inverse kinematics equations to find the corresponding leg lengths \((l_1, l_2, l_3, l_4, l_5, l_6)\) that are based on the estimate \(E\).
3. Compare these values to the actual leg lengths (FPOS variables in the controller that correspond to the encoder readings on the motors) to get a vector of errors \(\vec{e}_l\). If \(\vec{e}_l\) is less than some threshold, then we are done. If not, then continue.
4. Calculate Jacobian matrix \(J\) (this is a matrix of derivatives needed to converge to a solution) and then invert to get \(J^{-1}\).
5. The new estimation \((E_1)\) is obtained by taking the original estimation and adding a factor \((E_1 = E - J^{-1} \times \vec{e}_l\)).
6. Go back to part 2 using the new estimation.

The programming details for this process, especially steps 4 and 5, require a lot of tedious calculations and are beyond the scope of this paper. This information is available by request from ACS Motion Control, Inc. technical support. For the purpose of this document, it is sufficient to realize that an iterative process is required to find a solution to the forward kinematic equations. This process can be implemented in an ACSPL+ buffer, which will run continuously (the specific application wrote for this system has a forward kinematic update rate of 5 ms). The continuously updated forward kinematic solution will then let the user know the actual value of the commanded variables, which for the hexapod are the \((x, y, z, \alpha, \beta, \gamma)\) positions. The entire process of calculating inverse and forward kinematics is shown in Figure 5.

**Figure 5 – Forward and Inverse Kinematics Using ACSPL+**
6. Conclusions

Kinematic equations are at the heart of many mechanical systems. The ability to easily and accurately implement these equations may determine whether or not a controller can be used in a certain application. In this document, a simple tool tip system was examined to gain familiarity with kinematic equations. Then, the relationship between APOS, RPOS, and FPOS was explored and discussed in the context of a system utilizing kinematic equations. Finally, a specific system (the 6 DOF parallel manipulator Hexapod) was examined to explore how inverse and forward kinematic equations are implemented with a SPiiPlus series controller. Inverse kinematics with the SPiiPlus can be continuously calculated with one ACSPL+ buffer program. The CONNECT function then uses the continuously updated output of that program to command the associated motors/actuators. This allows the user to work (i.e. command simple or coordinated movements, generate smooth 3rd order motion profiles, etc.) entirely with the ‘commanded’ variables, which for the hexapod are the platform x, y, z, roll, pitch, and yaw coordinates. Forward kinematic equations are calculated by another continuously running ACSPL+ program. The result of these calculations is access to the actual (measured) values of the ‘commanded’ variables. Not only is this useful for the user, it is also necessary for monitoring feedback position of commanded variables and initializing offsets when enabling an axis. For this system, forward kinematics equations were not available in closed form solution, so an iterative process was used. Whether the kinematic equations of system are simple or complex, ACS Motion Control’s SPiiPlus controllers allow simple software implementation of these equations and the ability to generate accurate, coordinated motion profiles in the desired coordinate frame.