Simulation and Control of Robot Manipulators using Blind Adaptive Search Inverse Kinematics

Samuel N. Cubero
General Studies Department, Arts & Sciences, The Petroleum Institute
P. O. Box 2533, Abu Dhabi, United Arab Emirates (UAE)
scubero@pi.ac.ae and sam@samcubero.com

***Abstract* — This paper describes a simple, reliable general-purpose method for simulating and controlling all kinds of robot arms, manipulators and walking robot legs. Robotic manipulator arms are comprised of several solid links connected to each other in a serial chain, where each link only has one rotary or one translating (or sliding) joint, or a spherical joint. Called the 'Blind Adaptive Search Inverse Kinematics' (BASIK) method, this numerical IK method can control the end-effector (or tool) frame position and orientation for all types of complex manipulator designs, including those that have one or more internal singularities, redundant links or Jacobian matrices that cannot be inverted (i.e. where the 'Inverse Jacobian' method fails or where matrix inversion is impossible). The BASIK method can be implemented automatically within real-time animation and motion control software without the need to generate mathematical equations on paper or to determine the manipulator's Jacobian matrix.**

Keywords – inverse kinematics, robotics, robot, manipulator, simulation, control

I. INTRODUCTION

This paper briefly describes an easy-to-use, robust and general-purpose IK method that works for all types of serial-link manipulators, including robot arms with one or more redundant links and robot legs comprised of several serially connected solid links. Known as the 'Blind Adaptive Search Inverse Kinematics' (BASIK) method, this method can be easily implemented automatically by 3D simulation software to control all joint angles automatically without the need for complex derivations of mathematical or trigonometric equations and without the need to invert a Jacobian matrix. Only Forward Kinematic (FK) equations (or even just the standard Denavit-Hartenberg or D-H link parameters for all links) are required for the BASIK method to work, therefore, this method can be used to automatically calculate the IK solution (joint angles or linear displacements) for any type of robot arm design regardless of the number or type of links (or redundant links) or the kinds of singularities within the manipulator's workspace.

Robot motion control software was written, based on the BASIK method, allowing precise position and orientation control of the end-effector (or tool frame) for any custom designed manipulator or serial-link robot arm comprised of rotary, sliding or spherical joints (A spherical joint can be treated as one link having two rotary degrees of freedom rotating about one point in space). For position control, target position values are set (in 3D Cartesian coordinate format) and joint variables (angles or displacements) are automatically calculated using the BASIK method to guide and position the end-effector to the target position along a nearly straight-line path.

Along with gross 3D positioning of a wrist or end-effector tool, the orientation of the tool frame relative to the world coordinate frame axes can also be controlled using a numerical feedback control algorithm that searches for an IK solution which satisfies a maximum error tolerance value for both position error and the sum of all angular errors. Speed of convergence, solution stability and solution accuracy of the BASIK method depends on the initial values of various search parameters and the chosen tolerances for acceptable positional and angular errors (set by the user).

The next section describes the basic mechanical design of the VGTM (Variable Geometry Truss Manipulator) used for the leg of a walking robot. This is also referred to as the 'STIC Insect' robot leg. It was designed and built before the author had full knowledge of how to control it. Several different IK methods were tried but were unable to control this design reliably. The BASIK method is a radically simple numerical IK method that overcomes the limitations of popular IK methods described in several robotics textbooks and in several famous papers on this topic. This paper describes two popular IK methods and their inherent problems and limitations. A 3-link VGTM is used as a 'Case Study' manipulator because it is a difficult design to control using popular IK methods. The BASIK method is described in detail and is demonstrated using custom-made 3D simulation software that can assemble and control any kind of serial-link robot arm model.

John Billingsley (University of Southern Queensland, Australia), supervisor of the 'STIC Insect' walking robot project [1], suggested that tetrahedral pyramids be used for the construction of strong, lightweight limbs for the legs of a large walking robot, and this led to a new VGTM design.

II. VARIABLE GEOMETRY TRUSS MANIPULATOR VGTM

A Variable Geometry Truss Manipulator (VGTM) is a very lightweight yet strong and rigid form of serial-link robotic manipulator and one design will be used as a 'case study' to demonstrate the BASIK method. VGTM designs possess many advantages over other conventional mechanical designs used for building robot manipulators. VGTMs can be configured like multi-stage 'Stewart Platform' type (parallel) robots, where each 'link' is like a 6 DOF 'flight simulator' platform to position and orient a tool, like 'Tetrobot', made by Hamlin and Sanderson [2], [3], [4].

The most common and simplest building block (which forms one link) for a VGTM is a tetrahedral pyramid or a 6-member space frame. Such space frames may be used as static structures or they may serve as rotating links on a VGTM whereby each link rotates about one of its edges (typically a shaft or a rotary joint). Since neighboring joint axes (for rotary joints) of VGTMs are typically orthogonal to each other, this creates unsolvable mathematical and numerical problems if conventional or popular IK methods are employed to find the joint angles or displacements, due to the complexity of analytical equations. Such problems will be described in detail and solved later using BASIK.

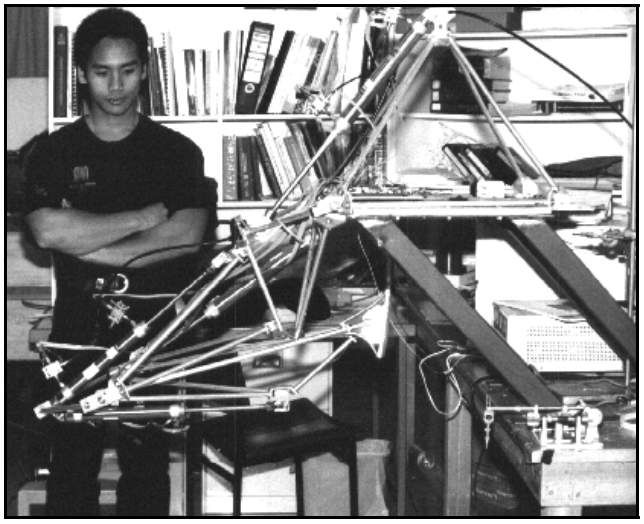

Figure 1. Three degree-of-freedom (3 DOF) VGTM robot leg curled up

The main advantages of VGTM (space frame) designs are:

- Very lightweight, minimal-material structures with high structural strength and rigidity (high stiffness). Low risk of structural damage due to low stresses / shared loads.
- Easy scalability of designs (very small to very large scale robots can be built based on the same design)
- Easier maintenance and repair (due to a high level of design modularity and easy replacement of components; One linear actuator used per link already has end-stops)
- Good controllability and stability even at high speeds of operation (due to very low mass or inertia for each link) especially if fast (ball-screw) linear actuators are used.
- Low materials and fabrication costs due to a simple design requiring few components and moving parts (per axis, or per Degree Of Freedom - DOF)

Figs. 1 and 2 show a 3 DOF robot leg in two different positions, namely, fully curled and fully extended. Without the steel air (pneumatic) cylinders attached, the VGTM leg structure weighs approximately 3 kg, yet the entire structure is very rigid and produces negligible deflections when loaded with forces of up to 300 N at the nodes. The nodes are the solid vertices which are made up of thick metal connection plates having holes, through which the ends of the long connection rods (solid or hollow tubes) are fastened with threaded nuts to form a rigid joint at the end of each rod. Each shaft is supported by 2 deep-groove ball bearing units.

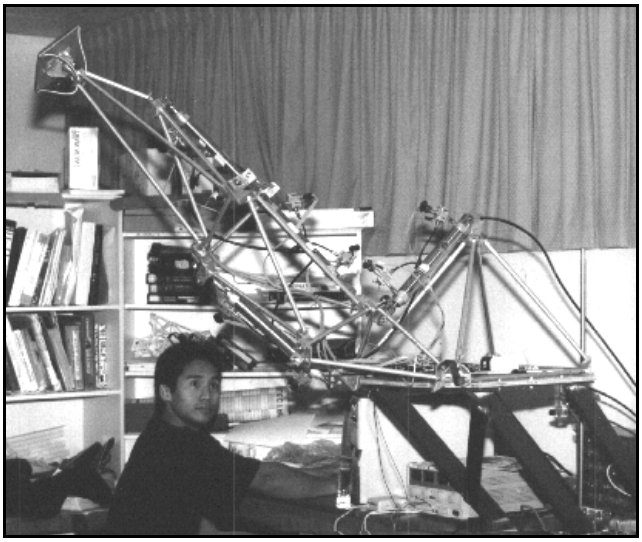

Figure 2. VGTM robot leg fully extended

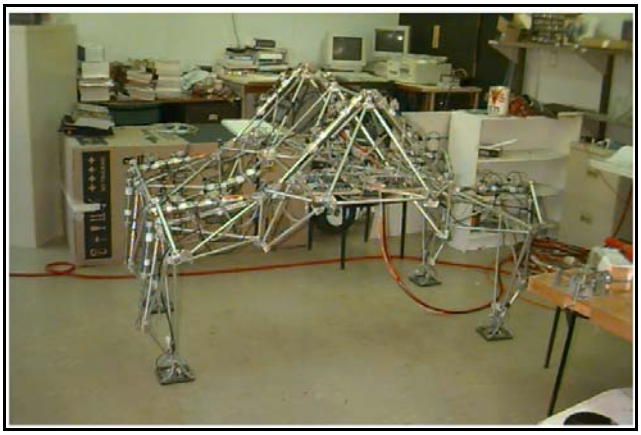

Figure 3. STIC Insect 4-legged air-powered walking robot built at USQ

Fig. 3 shows the 4-legged STIC Insect air-powered walking robot that was designed and built by the author at the University of Southern Queensland (USQ), Toowoomba,

Australia. STIC is an acronym for 'Space Truss Integrated Construction'. This robot was designed to be a rough terrain vehicle for walking over very uneven ground and boulders. The first prototype was powered by force and position controlled air cylinders using 48 experimental 'Floating Plate Gas Valves' (FPGVs) [5]. This robot was only able to walk over level ground but its walking performance was wobbly and prone to instability due to problems with the FPGVs.

III. Wireframe Graphics and Software Design

Microsoft™ Visual Basic (ver. 1.0) for MS-DOS™ was used to create a 3D wireframe simulation program using simple point-to-point 2D line drawing commands to display and erase lines and points on a 2D graphics display area. Three-dimensional points representing the vertices of all space frame links, described in each link's coordinate frame, were transformed to Cartesian coordinates of the 'World' coordinate frame in the model world. This was achieved by pre-multiplying vertex points (each represented as a 4x1 column vector) with one or more 4x4 homogeneous transformation matrices (or standard A matrices) from each link coordinate frame to the main 'World' coordinate frame (which is located at the center of the main robot body). The world coordinates for these vertices were then transformed to 2D screen coordinates using a 3D to 2D transform equation so that simple 2D lines can be drawn to join all the 2D points on the screen to create a pictorial style 3D wire-frame representation or view of all robot links on the computer screen based on the current joint angles. The Graphical User Interface (GUI) for this software is shown in Fig. 4.

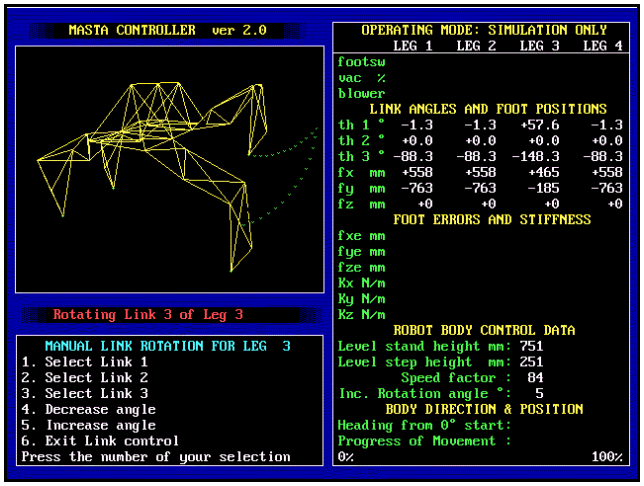

Figure 4. 3D wireframe simulation software for the STIC Insect robot

This software is used to display joint variable (angle) data, current foot positions, operating parameters (such as stepping height, body height, stiffness settings), progress details (a real-time progress bar), keyboard controls to drive the STIC Insect vehicle, 3D camera controls and 3D graphics. This kind of information needs to be visible or easily accessible to a robot operator. An information-rich user interface is important for real-time navigation and guidance control, checking and monitoring all operating variables (useful for problem solving or trouble-shooting), offline control programming (without hardware) and visualizing and verifying automatic walking gaits or walking styles prior to hardware control. Each leg is controlled like a 3 DOF robot manipulator where each foot must be controlled accurately in 3D world space to implement several different styles of walking patterns or 'gaits'. [1]

Fig. 5 shows a subroutine for drawing all the connection members (space frame links) of a single robot link. The 4 node points for each tetrahedral pyramid are recorded for each link and are expressed relative to each link's coordinate frame (usually located at the front end of the link, as will be described in a later section). In order for each wireframe line to be drawn correctly in the 2D graphics display area, each 3D point for a vertex (x, y, z point) in model space had to be 'transformed' using the appropriate number of A matrices (4x4 transformation matrices) so that it can be described as a 3D point relative to the 'World' coordinate frame. The variable 'Vaccum' in the program is a 4x4 'View Accumulation' matrix, which is the product of all transformation (or A) matrices multiplied together to give the *total transformation matrix* relating a particular link's frame to the 'World' frame. The two shaded lines of code in Fig. 5 transforms the first 3D point in 'World' space to a corresponding point in a 2D graphics display screen, as an (x, y) coordinate, or (pt2D.x, pt2D.y). At the end of this subroutine, lines are drawn in a 'point to point' manner so that each connection rod joining two vertices is drawn on the screen as a straight line, very quickly, for each link. 'Back face culling' methods [6] can be used to hide lines of triangles/surfaces that are not visible, like in Figs. 17 and 18.

```
SUB drawlinks (a AS INTEGER, b AS INTEGER, col AS INTEGER)

' a = leg number, b = link number, col = colour, n = node number, p = xyz1

DIM i AS INTEGER, j AS INTEGER, n AS INTEGER
DIM stnode AS INTEGER, Lpt AS INTEGER, nextnode AS INTEGER
DIM prod AS SINGLE, denom AS SINGLE

IF graphicsflag = 0 THEN EXIT SUB

' This subroutine draws all the lines of each link b on leg a due to a change
' in leg(a).link(b).theta due to manual control

' 1. First calculate the leg(a).link(b).Vaccum() matrix for this link
' 2. All node coordinates are transformed to 3D viewing coords, .vnode(n).pt(p)
' 3. leg(a).link(b).vnode(Llorder(0)),pt(xyz1) is projected to 2D as PSET
' 4. Point to point line drawing using Llorder() joins all 2D projected
'    points on the fly (directly converts each 3D point to 2D as each point
'    is required for continuation of the polyline, so that no time is wasted
'    in storing 2D coordinates in an array and recalling them again)

' Step 1. Calculate leg(a).link(b).Vaccum() matrix for this link
CALL Vmat(a, b)

' Step 2. Transform link node points to 3D points relative to the view frame
FOR n = 1 TO 4      ' Node loop
 FOR i = 1 TO 4     ' Row loop
  leg(a).link(b).vnode(n).pt(i) = 0  ' Initialise summing variable
  FOR j = 1 TO 4    ' Column loop for multiplying .Vaccum(i,j) with .node(n).pt(j)
   prod = leg(a).link(b).vaccum(i, j) * leg(a).link(b).node(n).pt(j)
   leg(a).link(b).vnode(n).pt(i) = leg(a).link(b).vnode(n).pt(i) + prod
  NEXT j
 NEXT i
NEXT n

' Step 3. PSET first point on the screen, using point order Llorder(0)
stnode = Llorder(0)      ' Starting node
denom = vdist - leg(a).link(b).vnode(stnode).pt(3)
pt2D.x = leg(a).link(b).vnode(stnode).pt(1) * vdist / denom
pt2D.y = leg(a).link(b).vnode(stnode).pt(2) * vdist / denom
IF b = 3 THEN CIRCLE (pt2D.X, pt2D.y), 5, 10   ' bright green path

PSET (pt2D.X, pt2D.y), col     ' Draw a point

' Step 4. Point to point line drawing using ordered Llorder(Lpt) Link points
FOR Lpt = 1 TO 5     ' Body point loop
  nextnode = Llorder(Lpt)
  denom = vdist - leg(a).link(b).vnode(nextnode).pt(3)
  pt2D.X = leg(a).link(b).vnode(nextnode).pt(1) * vdist / denom
  pt2D.y = leg(a).link(b).vnode(nextnode).pt(2) * vdist / denom
  LINE -(pt2D.X, pt2D.y), col     ' Draw line to this new point
NEXT Lpt

END SUB
```

Figure 5. Subroutine for displaying a 3D 'wireframe' image of a leg link

The link drawing subroutine is called several times by other subroutines that can modify the camera position or change a joint variable that affects link 'b'. It only needs to be called when the user's camera position changes or when a new link position needs to be displayed. The 'col' variable can be set to a yellow colour for displaying visible connection rods, or it can be set to 'black' (the background colour) for the purpose of erasing previously drawn lines. This allows the graphics screen to be updated much faster than clearing the screen for each display update.

Keyboard buttons were used to make selections from on-screen menus, access all available features and change all the possible control variables and operating modes for the robot. Fig. 4 shows how one link can be selected and controlled (rotated) with a changing 'theta' angle value. The joint variable data and 3D graphics are both automatically updated to display the current joint angles and link positions. A modern Windows™-style user interface could be developed for this simulation software, however, Microsoft Windows™ must typically share the main processor (CPU) with several other background applications (or different threads) which run in parallel with the simulation / control software. This may cause unexpected delays, unpredictable performance or unreliable timing issues, especially if the PC is used to send commands directly to several independent microcontroller chips (where each 8-bit microcontroller, or embedded controller, is responsible for servo-controlling or automatically adjusting the position or force for each actuator or motor). Hence, the MS-DOS™ operating system and MS Visual Basic™ for DOS (ver 1.0) were chosen for the task of controlling eight 8-bit Motorola™ HC11 (or HC811E2FN1) microcontroller chips, each of which controls the positions, stiffness settings and forces of two different air cylinders on a VGTM leg. There were a total of 4 legs, 16 air cylinders and 8 HC11 microcontrollers used on the STIC Insect robot. The data structures in the software had to be set up in such a way so all joint angles, control variables, stiffness settings and force settings could be easily accessed and modified with little effort. Several 'User-defined Data Structures' and 'TYPE variables' were used to store and retrieve all the information relating to each robot link. This allows the programmer to access all link variable values using the convenient format shown below.

`leg(1).Link(2).vnode(3).pt(1)=` (x-coordinate of node 3)
`leg(4).Link(3).Theta =` (Joint angle θ_3 for Leg 4, Link 3)

In essence, such a data structure is similar to a 'multidimensional array', however, user-defined names are much easier to read and use than numerically indexed arrays. Also, different data types (eg. INTEGER, SINGLE and DOUBLE precision floating-point numbers, even STRING variables) can be used within the same user-defined data structure, unlike numerically indexed multi-dimensional arrays which usually allow only one data type to be used for all variables or all elements in the entire array.

A human operator can automatically control a variety of walking motions for the STIC Insect robot. The operator can also manually control individual link rotations and point-to-point foot movements for any of the robot's legs. The diagram in Fig. 6 shows the main sections of the `MASTA.BAS` software (Master for Automatic Surface Transition Adaptation). Data initialization, gait initialization, serial communications, inverse kinematics, graphics and drawing routines were omitted from Fig. 6 for simplicity, however, these are called (when necessary) by subroutines or functions that need them (indicated by SUB). Subroutines which control the walking motions for the robot or those that modify joint angles have a name that starts with a capital '`M`', such as '`Minsect`', '`Mreptile`' and '`Mlinkrotate`'. Double-headed arrows mean the user can 'move' in both directions from one menu (or operating mode) to another, and vice versa. Single-headed arrows indicate 'one way' movements.

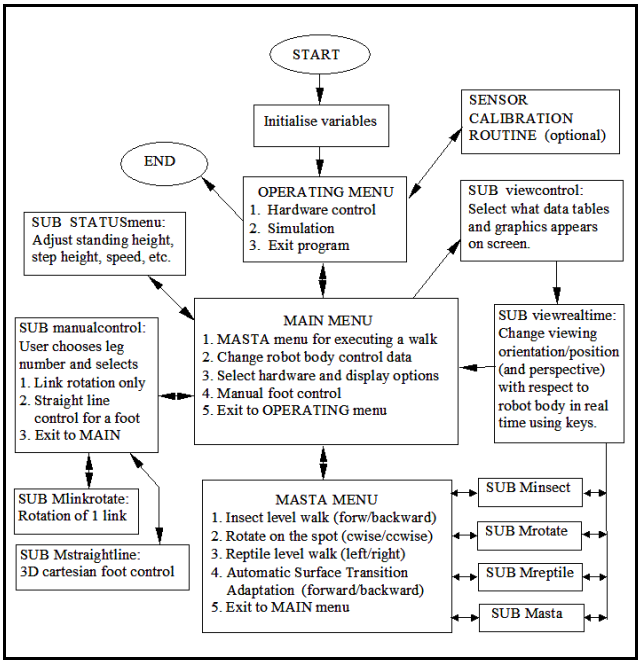

Figure 6. STIC Insect simulation and control software (called 'MASTA')

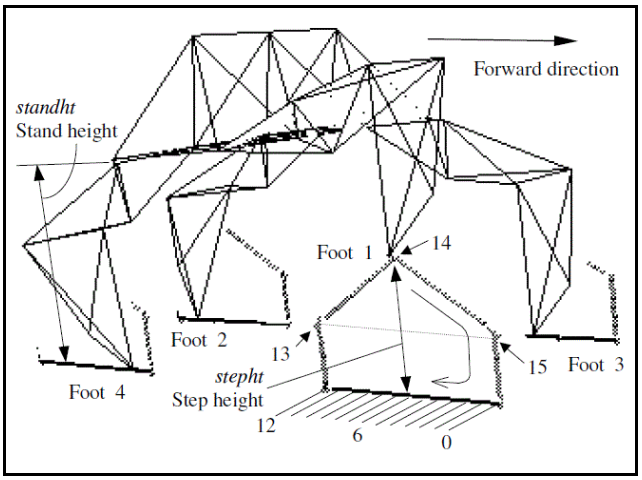

Figure 7. 'Insect' walking mode for forward (or backward) walking

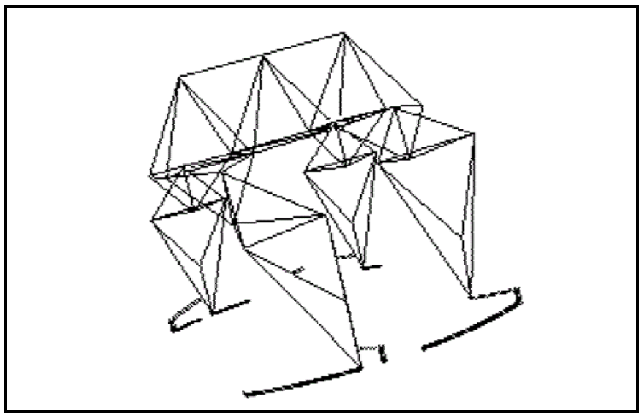

Figure 8. 'Rotation' mode for clockwise or anti-clockwise turning

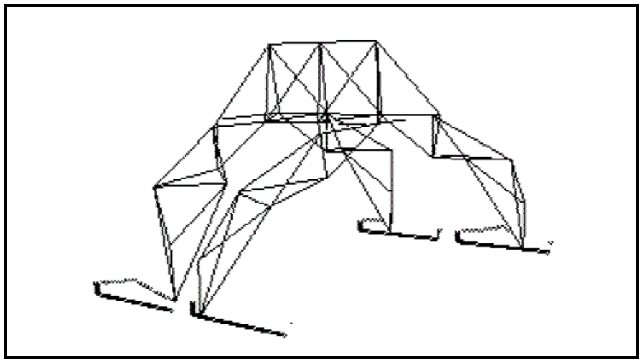

Figure 9. 'Reptile' mode for sidestepping in the left or right direction

Figs. 7, 8 and 9 show how 3D foot positions for the STIC Insect can be controlled accurately using the MASTA control software. Foot positions for all walking modes were all preprogrammed and played back (in a preset order) after all feet are placed in an 'initial starting position'. The following sections describe how 'point to point' movements can be achieved using Forward Kinematics (FK) and Inverse Kinematics (IK) methods. Two popular IK methods are also presented and discussed along with their inherent problems.

IV. Forward Kinematics for the STIC Insect Leg

Given the necessary link parameters and joint angle (or displacement) for each link of a manipulator, the position and orientation of the end-effector may be determined relative to the base coordinate frame (or 'reference frame') of the manipulator. This is called the Forward Kinematic (FK) solution and it can be found using A-matrix transforms, vectors or a direct geometric (or trigonometric) method. [7]

The *Denavit-Hartenberg* (D-H) *convention* defines a general procedure for assigning coordinate frames to the links of an articulated-link manipulator. The D-H method is described in several texts [7], [8], [10]. Using the D-H convention to assign coordinate frames allows a robot designer to systematically derive the transformation matrices that define the relationships between coordinate frames of neighboring links for any type of serial-link manipulator.

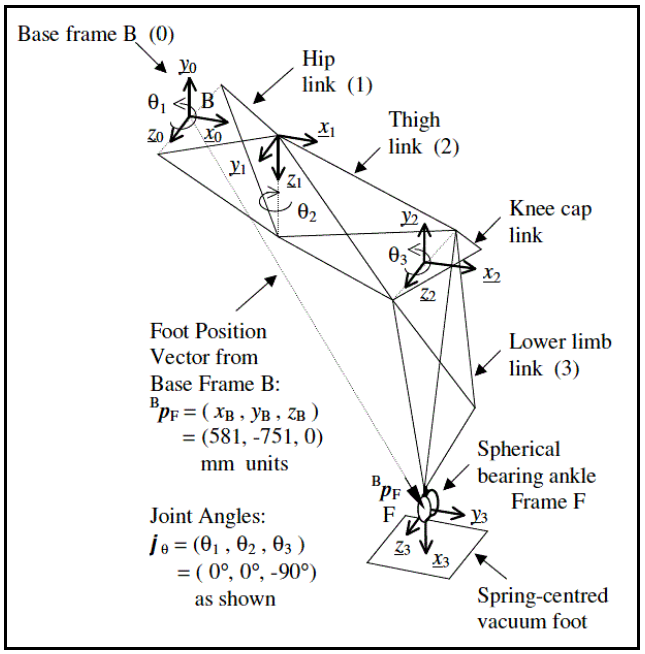

Figure 10. Right handed coordinate frames assigned using D-H convention

Such transformation matrices are called *A-matrices* and these depend on the *link parameters*, which are geometric variables unique to each link and its joint type (either rotary or translational). The A-matrix describes the geometric relationship between link frame *n* and link frame *n*-1. In general, for rotary links, the coordinate frame of a link should be located at the end of the link (farthest from the base when all links are fully extended) and the x-axis points towards the next coordinate frame. The entire link (or the link's coordinate frame) has one degree of freedom, and if it is rotary, the joint angle ('theta' value θ) is measured about the z axis of the previous coordinate frame which is closer to the base frame. For example, in Fig. 10, the joint angle for Link 2 is θ_2 , and this is measured about the z_1 axis.

TABLE I. Denavit-Hartenberg (D-H) Link Parameters

link *n*	θ_{min} deg°	θ_{max} deg°	θ_{max} - θ_{min} range, deg°	α_n deg°	l_n mm	d_n mm
1	-23°	67°	90°	90°	190	0
2	-33°	33°	66°	-90°	369	204
3	-170°	-8°	162°	0°	547	0

The D-H convention is used for assigning coordinate frames for each link of the STIC robot leg. The link parameters α_n , l_n and d_n are geometric constants described by McKerrow [7]. The general A-matrix transform is derived from the following rotations and translations [8]. This transformation matrix relates a link frame to the previous link frame (which is closer to the base frame). (NOTE: 'Rot' = Rotation transform, 'Trans' = Translation)

$$^{n-1}\mathbf{A}_n = \mathbf{Rot}(z, \theta)\ \mathbf{Trans}(l_n, 0, d_n)\ \mathbf{Rot}(x, \alpha) \quad (1)$$

$$^{n-1}\mathbf{A}_n = \begin{bmatrix} C\theta & -S\theta & 0 & 0 \\ S\theta & C\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & l_n \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha & -S\alpha & 0 \\ 0 & S\alpha & C\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

$$^{n-1}\mathbf{A}_n = \begin{bmatrix} C\theta_n & -S\theta_n C\alpha_n & S\theta_n S\alpha_n & l_n C\theta_n \\ S\theta_n & C\theta_n C\alpha_n & -C\theta_n S\alpha_n & l_n S\theta_n \\ 0 & S\alpha_n & C\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

The general A-matrix for a link with one rotary degree of freedom is given in (3) (n = link number, C = cos and S = sin). This transformation matrix is a function of the joint variable θ_n only, since each of the other link parameters, α_n , l_n , and d_n , are constant for a rigid link. The coordinate frames assigned to each leg of the STIC Insect walking robot are shown in Fig. 10 and the link parameters for the prototype robot are given in Table 1. The θ_{max} and θ_{min} values represent the maximum and minimum joint angles for a link, where (θ_{max} - θ_{min}) is the angular range of motion.

The 'knee cap' link in Fig. 10 is not assigned a coordinate frame or any link parameters since its purpose is to extend the range of motion of link 3, or the 'lower limb link', relative to link 2. Its angle relative to link 2 may be controlled if joint torque about the knee axis ($\underline{z}_2$ axis) must be controlled accurately based on piston force control. In such a case, a coordinate frame with extra link parameters may be necessary. The 'knee cap link' represents an extra *degree of mobility* but does not add a different or extra *degree of freedom* to the foot or end-effector. Its joint angle relative to link 2 does not affect the position of the robot foot unless one of its two attached linear actuators reaches an end-stop, therefore, the knee cap link can be ignored.

Using (3) and Table 1, we obtain the A-matrices for the links of the STIC Insect robot leg. In the following equations, a subscript following an 'S' or a 'C' represents the link number for the joint angle, θ, where S represents the 'sine' function and C represents the 'cosine' function (eg. $S_1 = \sin\theta_1$, $C_3 = \cos\theta_3$, etc.). The A matrices for the STIC robot leg are $^0\mathbf{A}_1$ (link 1), $^1\mathbf{A}_2$ (link 2) and $^2\mathbf{A}_3$ (link 3).

$$^0\mathbf{A}_1 = \begin{bmatrix} C_1 & 0 & S_1 & l_1C_1 \\ S_1 & 0 & -C_1 & l_1S_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textit{(relates frame 1 to frame 0)} \quad (4)$$

$$^1\mathbf{A}_2 = \begin{bmatrix} C_2 & 0 & -S_2 & l_2C_2 \\ S_2 & 0 & C_2 & l_2S_2 \\ 0 & -1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

$$^2\mathbf{A}_3 = \begin{bmatrix} C_3 & -S_3 & 0 & l_3C_3 \\ S_3 & C_3 & 0 & l_3S_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

These three transformation matrices may be combined to form the forward manipulator transform, $^B\mathbf{T}_F$, which describes coordinate frame 3 with respect to frame 0 (i.e. the transform relating foot frame, F, to the base frame, B).

$$^B\mathbf{T}_F = {}^0\mathbf{T}_3 = {}^0\mathbf{A}_1\ {}^1\mathbf{A}_2\ {}^2\mathbf{A}_3 \quad (7)$$

$$^B\mathbf{T}_F = \begin{bmatrix} C_1C_2C_3 - S_1S_3 & -C_1C_2S_3 - S_1C_3 & -C_1S_2 & C_1C_2(l_3C_3 + l_2) + S_1(d_2 - l_3S_3) + l_1C_1 \\ S_1C_2C_3 + C_1S_3 & C_1C_3 - S_1C_2S_3 & -S_1S_2 & S_1C_2(l_3C_3 + l_2) - C_1(d_2 - l_3S_3) + l_1S_1 \\ S_2C_3 & -S_2S_3 & C_2 & S_2(l_3C_3 + l_2) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8)$$

The first three columns of the FK solution matrix in (8) represent the unit vectors of the x, y and z axes of the 'foot frame' (Frame F), respectively, thus defining the exact orientation of Frame F relative to the base frame (B or 0). The forward kinematic solution for calculating the position of the foot (origin of frame F, or point F) with respect to base frame B is read directly as the last column in the matrix of (8), where $^B\boldsymbol{p}_F$ is the rightmost column vector of $^B\mathbf{T}_F$. The Forward Kinematic or FK equations which define the position of the robot 'foot' F (or the centre of the spherical bearing 'ankle') relative to the base frame B, are given by:

$$x_B = \cos\theta_1 \cos\theta_2 (l_3 \cos\theta_3 + l_2) + \sin\theta_1 (d_2 - l_3 \sin\theta_3) + l_1 \cos\theta_1 \quad (9)$$

$$y_B = \sin\theta_1 \cos\theta_2 (l_3 \cos\theta_3 + l_2) - \cos\theta_1 (d_2 - l_3 \sin\theta_3) + l_1 \sin\theta_1 \quad (10)$$

$$z_B = \sin\theta_2 (l_3 \cos\theta_3 + l_2) \quad (11)$$

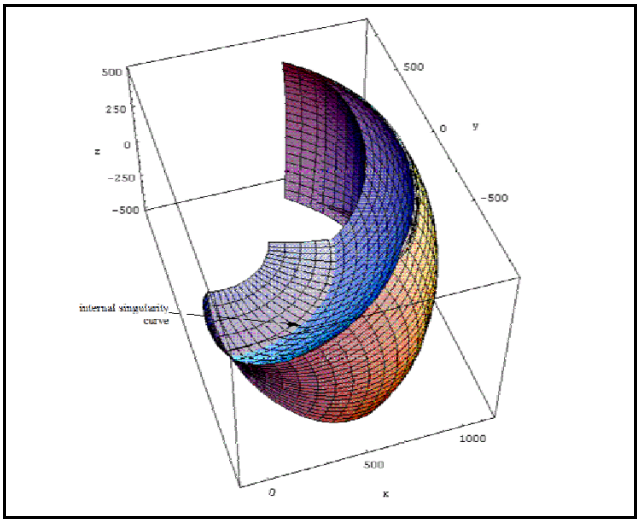

Figure 11. Workspace boundary for the VGTM leg of the STIC Insect

Equations (9) to (11) can be plotted using extreme angle positions (the θ_{min} and θ_{max} values for each link) to produce the extreme or outer 'workspace boundary' surfaces, as shown in Fig. 11. These surfaces represent the limits of reach for the 'ankle' of the robot leg, or the extreme positions that point F (the origin of Frame F) can reach. Any positions outside of the workspace volume are impossible for the foot (F) to reach. Such plots can be produced easily using commercial mathematical analysis software packages such as Mathematica™ , Maple™ and MATLAB™.

This model of the STIC Insect manipulator leg does not have any *redundant* links. i.e. There is only one joint angle solution (θ_1, θ_2, θ_3) for any given foot position (x_B, y_B, z_B). The only exception to this is when the end-effector (ankle or foot) lies on the planar curve where the left and right side surfaces of the workspace boundary intersect, as shown in Fig. 11. It is worthwhile noting that there is an '*internal singularity*' in the form of a curve inside the workspace of the leg. This curve represents a *degenerate configuration* because it is the locus of foot positions where any value of θ_2 can be used to achieve the same foot position (x_B, y_B, z_B), when θ_1 and θ_3 are constant. In mathematical terms, this curve is known as a *singularity*, where there are an infinite number of joint angle solutions for any given position of F along this curve. This internal singularity occurs when angle $\theta_3 \approx -132.4°$, or when the angle behind the knee joint, between links 2 and 3, is about 47.6°. When point F (the 'ankle') is on or near this curve, there are an infinite number of possible solutions for θ_2 , because point F coincides with the axis of z_1 . Such a *singularity* curve should be avoided (by not letting point F near such a curve) because most of the popular IK methods tend to fail or become unpredictable near such curves, as will be discussed in the next section.

V. Common Inverse Kinematics Methods

Almost all industrial revolute-joint robot arm controllers need to use an IK method (in software) to provide precise positioning and orientation control of tools or end-effector attachments in 3D Cartesian space. XYZ Gantry robots do not require IK unless rotating or revolute joints are used. Closed-form IK equations are used to calculate the joint angles (or joint displacements) for only a few classes of simple robots. In fact, it is impossible to derive or achieve closed-form or explicit equations to calculate joint variables for all kinds of robot arm designs, where the joint variable being calculated is completely independent of all the other joint variables. IK is important for finding the joint angles or displacements needed to achieve a desired frame position and orientation for a robot's end-effector or tool in 3D Cartesian-coordinate space, relative to a fixed frame like the 'World' frame or a Base frame 'B'. Unfortunately, neighbouring joint axes that are orthogonal to each other cause solution failures using popular IK methods (such as Paul's method [8] to obtain an explicit equation, or the popular 'Inverse Jacobian' method [7]). A walking robot also needs to accurately control the placement of its feet while walking over rough terrain so that suitable footholds can be attained while maintaining dynamic and static stability of its body, therefore, achieving an accurate IK solution quickly for each leg is very important for smooth and predictable walking performance.

The *general transformation matrix* which relates a coordinate frame N to a reference frame R is given by

$$ {}^{R}\mathbf{T}_{N} = \begin{bmatrix} a & d & g & p \\ b & e & h & q \\ c & f & i & r \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} {}^{R}\underline{x}_{N} & {}^{R}\underline{y}_{N} & {}^{R}\underline{z}_{N} & {}^{R}\boldsymbol{p}_{N} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12} $$

where

$$ {}^{R}\underline{x}_{N} = a\,\underline{x}_{R} + b\,\underline{y}_{R} + c\,\underline{z}_{R} \tag{13} $$

(unit vector of $\underline{x}_N$ axis with respect to frame R)

$$ {}^{R}\underline{y}_{N} = d\,\underline{x}_{R} + e\,\underline{y}_{R} + f\,\underline{z}_{R} \tag{14} $$

(unit vector of $\underline{y}_N$ axis with respect to frame R)

$$ {}^{R}\underline{z}_{N} = g\,\underline{x}_{R} + h\,\underline{y}_{R} + i\,\underline{z}_{R} \tag{15} $$

(unit vector of $\underline{z}_N$ axis with respect to frame R)

The above three column vectors define the directions of the coordinate frame axes of frame N with respect to frame R after the transformation is performed. The displacement vector from the origin of frame R (point R) to the origin of frame N (point N) is given by

$$ {}^{R}\boldsymbol{p}_{N} = p\,\underline{x}_{R} + q\,\underline{y}_{R} + r\,\underline{z}_{R} \tag{16} $$

In terms of roll, pitch and yaw angles, the general transform matrix relating frame N to frame R can also be expressed as:

$$ {}^{R}\mathbf{T}_{N} = \begin{bmatrix} C\phi\,C\theta & C\phi\,S\theta\,S\psi - S\phi\,C\psi & C\phi\,S\theta\,C\psi + S\phi\,S\psi & p_x \\ S\phi\,C\theta & S\phi\,S\theta\,S\psi + C\phi\,C\psi & S\phi\,S\theta\,C\psi - C\phi\,S\psi & p_y \\ -S\theta & C\theta\,S\psi & C\theta\,C\psi & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{17} $$

The roll, pitch and yaw angles (ψ, θ, and ϕ respectively) of frame N's axes with respect to frame R's axes may be solved directly from a given ${}^{R}\mathbf{T}_{N}$ transform [7].

$$ \text{Yaw:} \quad \phi = \text{atan2}\,(\,b,\,a\,) \tag{18} $$

$$ \text{Pitch:} \quad \theta = \text{atan2}\,(\,-c,\,a\,C\phi + b\,S\phi\,) \tag{19} $$

$$ \text{Roll:} \quad \psi = \text{atan2}\,(\,f,\,i\,) \tag{20} $$

A *closed-form algebraic solution* is an *explicit* equation for a joint variable and is the fastest and simplest format for calculating a joint variable. For example, the joint angle θ_1

$= f\,(x_B, y_B, z_B)$ is an explicit closed-form equation, where f is a function of the end-effector position, but to work, θ_1 must be independent of the other joint angles θ_2 and θ_3.

A. *Paul's IK method*

The following method may be used in an attempt to solve for explicit expressions for the joint variables of the STIC robot leg. It is based on Paul's method of matrix manipulation [7], [8]. Firstly, we equate the general transformation matrix, ${}^{R}\mathbf{T}_{N}$, of (17) to the forward manipulator transformation matrix, ${}^{B}\mathbf{T}_{F}$, of (8). We are interested in expressions which involve p_x , p_y , p_z , or the foot position, (x_B, y_B, z_B). We look at the right-most column vector of (8) and form the forward kinematics equations, (9)-(11). Secondly, we attempt to solve for one joint variable at a time by isolating each joint variable. This may be done by obtaining an explicit 'sin' function and 'cos' function for the joint variable of interest, dividing the 'sin' function by the 'cos' function to obtain an explicit 'tan' function and finally, solving for the joint variable using the 'atan2' function (which returns an angle between 0°-360°). Unfortunately, for the STIC robot leg design, this cannot be done because $\cos\theta_1$, $\sin\theta_1$, $\cos\theta_2$, $\sin\theta_2$, $\cos\theta_3$ and $\sin\theta_3$ functions cannot be defined independently of each other, i.e. each joint angle cannot be expressed independently of the other joint angles. New sets of equations for the joint angles may be created by pre-multiplying the left and right hand sides of (7) by the inverse of ${}^{0}\mathbf{A}_{1}$ so that

$${}^{0}\mathbf{A}_{1}^{-1}\,{}^{B}\mathbf{T}_{F} = {}^{1}\mathbf{A}_{2}\,{}^{2}\mathbf{A}_{3} \tag{21}$$

Even when these matrices are equated, a 'sin' and 'cos' function for a joint angle still cannot be uniquely defined for creating an 'atan2' function. Again, we may generate a new set of equations by pre-multiplying both sides of (21) with ${}^{1}\mathbf{A}_{2}^{-1}$ in another attempt to find explicit equations.

$${}^{1}\mathbf{A}_{2}^{-1}\;{}^{0}\mathbf{A}_{1}^{-1}\,{}^{B}\mathbf{T}_{F} = {}^{2}\mathbf{A}_{3} \tag{22}$$

Again, we face the problem of being unable to express a joint angle as an explicit function of (x_B , y_B , z_B) and the link parameters only, independent of the other joint angles. Hence, there is *no* explicit algebraic closed-form inverse-kinematics solution for the STIC Insect manipulator of Fig. 10. This problem is related to the fact that the joints of all neighboring links are orthogonal to each other. Trigonometric functions tend towards zero or infinity at 0 or 90 degree positions. Paul's method of IK fails to provide closed-form solutions for the three joint angles (θ_1 , θ_2 , θ_3) of the STIC Insect leg, because each joint angle is related to one or more other joint angles, as shown in (9), (10) and (11), and each joint angle cannot be individually defined as an explicit function independent of any other joint angle.

B. *Inverse Jacobian IK method*

The 'Inverse Jacobian' Inverse Kinematics (IK) method can be used to calculate joint angles / variables quickly (in real time) to achieve a desired foot position [7]. This method of IK is one of the oldest and most popular methods for achieving an IK solution, but it can be unreliable and may fail to find a solution. An inverse kinematics solution may be obtained by inverting the manipulator's *Jacobian* matrix, either symbolically, or numerically. The Jacobian is a special type of square transformation matrix which relates all incremental joint angle changes, $\Delta\boldsymbol{j}_\theta = (\Delta\theta_1, \Delta\theta_2, \Delta\theta_3)$, to the end-effector's incremental movements in cartesian space, $\Delta^{B}\boldsymbol{p}_F = (\Delta x_B, \Delta y_B, \Delta z_B)$, relative to the base frame B. The Jacobian matrix for the STIC Insect robot leg of Fig. 10 is

$${}^{B}\mathbf{J}_{\theta} = \begin{bmatrix} \frac{\partial x_B}{\partial \theta_1} & \frac{\partial x_B}{\partial \theta_2} & \frac{\partial x_B}{\partial \theta_3} \\ \frac{\partial y_B}{\partial \theta_1} & \frac{\partial y_B}{\partial \theta_2} & \frac{\partial y_B}{\partial \theta_3} \\ \frac{\partial z_B}{\partial \theta_1} & \frac{\partial z_B}{\partial \theta_2} & \frac{\partial z_B}{\partial \theta_3} \end{bmatrix} \tag{23}$$

where

$$\frac{\partial x_B}{\partial \theta_1} = -S_1 C_2 (l_3 C_3 + l_2) + C_1 (d_2 - l_3 S_3) - l_1 S_1 \tag{24}$$

$$\frac{\partial x_B}{\partial \theta_2} = -C_1 S_2 (l_3 C_3 + l_2) \tag{25}$$

$$\frac{\partial x_B}{\partial \theta_3} = - l_3 C_1 C_2 S_3 - l_3 S_1 C_3 \tag{26}$$

$$\frac{\partial y_B}{\partial \theta_1} = C_1 C_2 (l_3 C_3 + l_2) + S_1 (d_2 - l_3 S_3) + l_1 C_1 \tag{27}$$

$$\frac{\partial y_B}{\partial \theta_2} = -S_1 S_2 (l_3 C_3 + l_2) \tag{28}$$

$$\frac{\partial y_B}{\partial \theta_3} = l_3 C_1 C_3 - l_3 S_1 C_2 S_3 \tag{29}$$

$$\frac{\partial z_B}{\partial \theta_1} = 0 \tag{30}$$

$$\frac{\partial z_B}{\partial \theta_2} = C_2 (l_3 C_3 + l_2) \tag{31}$$

$$\frac{\partial z_B}{\partial \theta_3} = - l_3 S_2 S_3 \tag{32}$$

where $C_i = \cos\theta_i$ and $S_i = \sin\theta_i$ (shorthand notation).

The number of rows in the Jacobian matrix of (23) equals the number of degrees of freedom, in cartesian space, which are needed for the end-effector to perform a task. For example, a STIC Insect robot leg must position foot F to a point (x_B, y_B, z_B) in its cartesian workspace, hence, only

three rows are needed for its Jacobian. The number of columns in the Jacobian matrix is determined by the number of joints in the serial-link mechanism. As mentioned earlier, it is convenient to ignore the extra degree of mobility provided by the 'knee cap link' (Fig. 10) since it shares the same degree-of-freedom as Link 3, hence, the Jacobian needs only three columns. If the Jacobian matrix has more columns than rows, the manipulator is said to be *redundant*.

The relationship between the incremental cartesian displacements of the end-effector (or point F), $\Delta^{B}\boldsymbol{p}_{F}$, and the incremental changes in joint angles, $\Delta\boldsymbol{j}_{\theta}$, is given by

$$\Delta^{B}\boldsymbol{p}_{F} = {}^{B}\mathbf{J}_{\theta}\,\Delta\boldsymbol{j}_{\theta} \tag{33}$$

or, in matrix form:

$$\begin{bmatrix}\Delta x_B\\ \Delta y_B\\ \Delta z_B\end{bmatrix} = \begin{bmatrix}\frac{\partial x_B}{\partial \theta_1} & \frac{\partial x_B}{\partial \theta_2} & \frac{\partial x_B}{\partial \theta_3}\\ \frac{\partial y_B}{\partial \theta_1} & \frac{\partial y_B}{\partial \theta_2} & \frac{\partial y_B}{\partial \theta_3}\\ \frac{\partial z_B}{\partial \theta_1} & \frac{\partial z_B}{\partial \theta_2} & \frac{\partial z_B}{\partial \theta_3}\end{bmatrix}\begin{bmatrix}\Delta\theta_1\\ \Delta\theta_2\\ \Delta\theta_3\end{bmatrix} \tag{34}$$

Therefore, the *differential inverse kinematics solution* for all incremental joint angles is obtained by premultiplying both sides of (34) by the inverse of the Jacobian. Fortunately, the Jacobian for the STIC Insect leg is square, so it is easily invertible (except when F is at a singularity).

$$\Delta\boldsymbol{j}_{\theta} = {}^{B}\mathbf{J}_{\theta}{}^{-1}\;{}^{B}\Delta\boldsymbol{p}_{F} \tag{35}$$

$$\begin{bmatrix}\Delta\theta_1\\ \Delta\theta_2\\ \Delta\theta_3\end{bmatrix} = \begin{bmatrix}\frac{\partial x_B}{\partial \theta_1} & \frac{\partial x_B}{\partial \theta_2} & \frac{\partial x_B}{\partial \theta_3}\\ \frac{\partial y_B}{\partial \theta_1} & \frac{\partial y_B}{\partial \theta_2} & \frac{\partial y_B}{\partial \theta_3}\\ \frac{\partial z_B}{\partial \theta_1} & \frac{\partial z_B}{\partial \theta_2} & \frac{\partial z_B}{\partial \theta_3}\end{bmatrix}^{-1}\begin{bmatrix}\Delta x_B\\ \Delta y_B\\ \Delta z_B\end{bmatrix} \tag{36}$$

A basic *Inverse Jacobian method* for controlling the end position of the robot leg in Fig. 10 (or any serial-link manipulator) is demonstrated by the following algorithm.

1. Set the scalar trajectory step, distance *s* (in mm) for differential motion of the foot (*s* can be fixed or scaled as a fraction of the distance to the next goal point). Initialize or clear Error Flag *ef* = 0.
2. Initialise target foot position ${}^{B}\boldsymbol{t}_{T}$ (point T) to equal the current foot position ${}^{B}\boldsymbol{p}_{F}$ (point F) and clear the foot position error vector, ${}^{F}\boldsymbol{e}_{T} = (0, 0, 0)$.
3. Trajectory planning algorithm defines the next or demanded target foot position ${}^{B}\boldsymbol{t}_{T} = (tx_B, ty_B, tz_B)$ which is normally within a distance less than 2*s* from the last ${}^{B}\boldsymbol{p}_{F}$ position. (This algorithm takes note of the *ef* error flag also and may modify the trajectory if *ef* = 1) Clear error flag, set *ef* = 0.
4. Calculate the current foot position ${}^{B}\boldsymbol{p}_{F} = (x_B, y_B, z_B)$ with the FK equations (9), (10) & (11).
5. Calculate the foot position error vector which needs to be minimized: ${}^{F}\boldsymbol{e}_{T} = {}^{B}\boldsymbol{t}_{T} - {}^{B}\boldsymbol{p}_{F}$
6. Set the desired relative incremental displacement vector for the foot:
   $\Delta^{F}\boldsymbol{s}_{T} = (sx_F, sy_F, sz_F) = s\,{}^{F}\underline{u}_{T} = s\,(a_F, b_F, c_F)$, where ${}^{F}\underline{u}_{T} = (a_F, b_F, c_F)$ is the unit vector in the direction of travel from point F to T (the target point T).
7. Calculate the modified differential foot displacement vector using the formula $\Delta^{F}\boldsymbol{p}_{N} = \Delta^{F}\boldsymbol{s}_{T} + {}^{F}\boldsymbol{e}_{T}$ (When this vector is added to ${}^{B}\boldsymbol{p}_{F}$, the resulting vector will place the foot at some point N that is very close to the target, T).
8. Attempt to invert the Jacobian and obtain ${}^{B}\mathbf{J}_{\theta}{}^{-1}$ numerically using Gaussian elimination (or a similar matrix inversion method). If the matrix is singular (i.e. foot is outside workspace or at the internal singularity), there is no solution so set the error flag, *ef*=1. In this case, choose a valid target position and return to Step 3.
9. Calculate the differential joint angle solution for the leg, $\Delta\boldsymbol{j}_{\theta} = {}^{B}\mathbf{J}_{\theta}{}^{-1}\,\Delta^{F}\boldsymbol{p}_{N} = (\Delta\theta_1, \Delta\theta_2, \Delta\theta_3)$.
10. Test the new joint angles using $\boldsymbol{jt}_{\theta} = (\theta_1 + \Delta\theta_1\,,\ \theta_2 + \Delta\theta_2\,,\ \theta_3 + \Delta\theta_3)$. If any link angle lies outside of its acceptable range of motion, do not use this solution, set the error flag *ef* =1 , go to Step 3.
11. If the error flag is clear, *ef* = 0, then update the joint angles, $\boldsymbol{j}_{\theta} = \boldsymbol{jt}_{\theta}$, and send this data to the closed-loop (PD) position controllers of the link actuators. If *ef* = 1, then the current joint angles are not changed and a new target must be chosen.
12. Return to Step 3. (*End of control loop*)

When calculating the inverse Jacobian numerically using Gaussian elimination, small computational errors arise in the solutions for the new joint angles, $\boldsymbol{j}_{\theta} + \Delta\boldsymbol{j}_{\theta}$. These errors are due to inaccuracy of the previous joint angle solution and the finite machine precision error for digital numbers used in the calculation of the differential joint angle solutions. *Single precision* 32-bit (floating point type) numbers can be used to produce reasonably fast real-time solutions for joint angles on most modern PC's but with the limitation of only about 7 digits of numerical precision. The difference between the current foot position, ${}^{B}\boldsymbol{p}_{F}$, and the new target point T where the foot should be, ${}^{B}\boldsymbol{p}_{T} = {}^{B}\boldsymbol{t}_{T}$, is defined as the *position error vector*, ${}^{F}\boldsymbol{p}_{T} = {}^{B}\boldsymbol{t}_{T} - {}^{B}\boldsymbol{p}_{F} = {}^{F}\boldsymbol{e}_{T}$. This error vector gradually becomes large enough in magnitude to prevent a foot from 'backtracking' perfectly over a 3D path in space that it had previously travelled through, regardless of how small we make $\Delta^{B}\boldsymbol{p}_{F}$. Hence, it is impossible to accurately control the position of a foot in 'leg space' Cartesian coordinates (relative to base frame B) by only adding the solutions of (35) to $\boldsymbol{j}_{\theta}$ because the joint angle errors will always accumulate over time. To

minimize the growth of this calculation error, we must 'feed back' the actual current position of the foot and then modify the theoretical $\Delta^{B}\boldsymbol{p}_{F}$ to compensate for the effects of the error vector. This helps to minimize the effect of growing computational errors.

The 'Inverse Jacobian' inverse kinematics method allows for very smooth and fast real-time trajectory control with insignificant joint angle errors when using almost any modern PC computer (Intel™ 386 or faster processor built after 1995, running at a CPU clock speed of 50 MHz or faster). Foot position errors introduced by this algorithm are usually less than ± 0.1 mm for small point-to-point, absolute foot displacements of less than 10 mm (trajectory step *s*). Smoother and more accurate foot trajectory can be obtained by reducing the magnitude of the trajectory step, *s*, at the expense of lower real time control speeds, but this is not a problem with today's PCs that typically run at CPU clock speeds above 2 GHz. The trajectory step, *s*, is proportional to the real time instantaneous speed of the foot for a fixed computational speed. Trajectory smoothness and positional resolution and accuracy at high speeds can be further improved by executing the control software on faster computers and using smaller values for *s*.

Although being a very popular IK method that works for most robot arm designs, the 'Inverse Jacobian' IK method requires significant mathematical derivation work and programming effort, it can become unstable or unreliable at achieving a solution (especially when the end-effector is at a singularity) and it cannot be used for all types of serial-link robotic manipulators, especially those that have redundant links (because the non-square Jacobian matrices of such manipulators cannot be inverted). Figs. 7, 8 and 9 are actual 'screenshots' showing near-perfect foot positioning accuracy that can be achieved using this method. Joint angles were calculated very quickly and efficiently, however, whenever a foot position was too close to an 'internal singularity' or 'workspace boundary', the Jacobian matrix could not be inverted due to computational errors (e.g. division by zero errors cause failures or 'crashing' of the matrix inversion algorithm). Just like Paul's IK method, the 'Inverse Jacobian' IK method is also not easy to implement automatically in general-purpose simulation and control software due to solution failures and redundant links.

C. *Other IK methods*

There are several other different methods that can be used to obtain an IK solution, as published in various papers. These are described in detail in several different textbooks like [7] and [10], however discussing all of these other methods in detail is beyond the scope of this paper. Using 'quaternions', simple vectors and trigonometric approaches can work, however, they typically require a great deal of complex mathematical derivation and equation solving effort, making them difficult to use or automate with software. Most IK methods also fail to find a solution at internal singularities, where there are an infinite number of solutions for a joint variable, or where there are more than one robot arm 'poses' or configurations that can be selected.

VI. BLIND ADAPTIVE SEARCH INVERSE KINEMATICS

After considering the limitations of popular IK methods, the author imagined that there had to be a simpler and more reliable way of obtaining an IK solution for any kind of robot arm design, without requiring the skills of an award-winning mathematics professor. The *Blind Adaptive Search Inverse Kinematics* (or BASIK) method was therefore developed to be a 'general purpose' IK method that overcomes virtually all of the common limitations and problems of other IK methods. It is amazingly simple, reliable and easy to use and is largely based on a 'trial and error' approach to solving the joint displacements (rotary or linear). This IK method finds all the correct joint variables needed which correspond to incremental changes in the displacements and/or axis orientations of a robot's end-effector frame in 3D space. For simplicity, the following section describes the BASIK method for position control only, but position and orientation control of an end-effector frame will be described later on.

A. *How the BASIK method works for position control only*

Consider a 3 Degree-Of-Freedom (or 3 DOF) serial-link manipulator with only rotary joints, such as the example robot leg in Figs. 10 or 12. Assuming that a small change in foot position (point F) is required, each joint variable (joint angle θ or displacement *d*) can only do one of the following:

- Increase (+) by a small displacement $\Delta\theta$ (rotary), or Δd (sliding)
- Decrease (-) by a small displacement $\Delta\theta$ (rotary), or Δd (sliding); or
- Remain the same (0) , i.e. no change

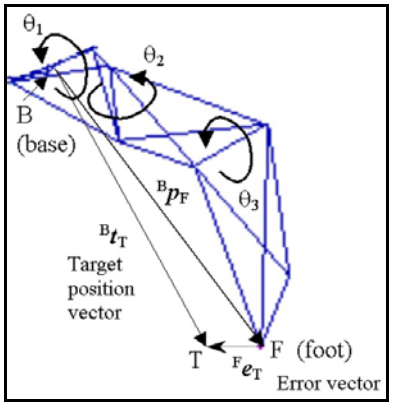

Figure 12. Position vectors for Foot position F and target point T

With FK equations, we can calculate the end-effector frame position error and/or basis axes alignment errors which arise for all possible combinations of small joint variable adjustments. For any serial-link manipulator with *n* degrees of freedom, there are a total of 3^n -1 different possible combinations for joint variable adjustments which can be made for a given $\Delta\theta$ (for rotary joints), or Δd for translational joints (ignoring the 'no changes' solution).

TABLE II. JOINT VARIABLE DISPLACEMENT COMBINATIONS FOR THE 3 DOF ROBOT LEG IN FIG. 10

Combinations for joint variables (index i)				***Positions***	***Error magnitudes***
Test 1: $\theta_{1t1} = \theta_1+\Delta\theta$, $\theta_{2t1} = \theta_2+\Delta\theta$, $\theta_{3t1} = \theta_3+\Delta\theta$ (+, +, +) (i = 1)				${}^{B}\boldsymbol{p}_{Ft}$ (i =1)	$\lvert{}^{F}\boldsymbol{e}_{Tt}\rvert$ (1)
Test 2: $\theta_{1t2} = \theta_1+\Delta\theta$, $\theta_{2t2} = \theta_2+\Delta\theta$, $\theta_{3t2} = \theta_3-\Delta\theta$ (+, +, -) (i = 2)				${}^{B}\boldsymbol{p}_{Ft}$ (2)	$\lvert{}^{F}\boldsymbol{e}_{Tt}\rvert$ (2)
Test 3: $\theta_{1t3} = \theta_1+\Delta\theta$, $\theta_{2t3} = \theta_2-\Delta\theta$, $\theta_{3t3} = \theta_3+\Delta\theta$ (+, -, +) (i = 3)				${}^{B}\boldsymbol{p}_{Ft}$ (3)	$\lvert{}^{F}\boldsymbol{e}_{Tt}\rvert$ (3)
+, -, - (i = 4)	+, 0, +	-, -, 0	+, 0, -	(array continues)	(array continues)
-, +, + (i = 5)	+, 0, 0	-, 0, -	+, -, 0	etc.	etc.
-, +, - (etc.)	0, +, +	-, 0, 0	-, 0, +		
-, -, +	0, +, 0	0, -, -	-, +, 0		
-, -, -	0, 0, +	0, -, 0	0, +, -	(test positions)	(test errors)
+, +, 0	*Ignore* 0, 0, 0	0, 0, -	0, -, + (i = 26)	${}^{B}\boldsymbol{p}_{Ft}$ (26)	$\lvert{}^{F}\boldsymbol{e}_{Tt}\rvert$ (26)

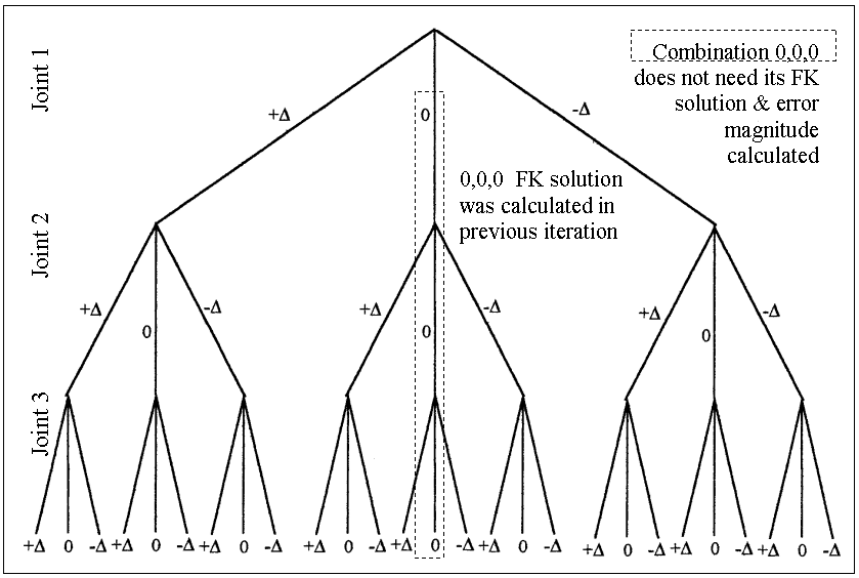

Figure 13. Search space for calculating the errors for all possible combinations of joint angle changes for the 3 DOF robot leg in Fig. 10 (for a given Δθ)

For example, a 2 link robot arm would have only 3^2-1 = 8 possible combinations (excluding 'no changes'). In the following discussion, we will consider solving the 3 joint angles for the example robot leg in Fig. 10, given the desired target position ${}^{B}\boldsymbol{t}_{T} = (x_B , y_B , z_B)$ for the foot relative to the base frame B. The following analysis can also be extended to 6 degree of freedom industrial robot arm manipulators, where the position of the end-effector (or tool) frame origin position and (basis) axes orientations are specified and must be achieved through small changes in joint variables. For simplicity, we will now discuss obtaining an IK solution for positioning an end-effector frame (origin point) at a desired (target) point. Later, we will discuss how to achieve an IK solution for both position and orientation control of the end-effector (or tool) frame, as defined in Cartesian space relative to a reference frame.

For the example robot in Fig. 10, there are N = 3^3-1 = 26 possible FK solutions for the foot position if each joint angle changes by +Δθ, -Δθ or 0° (no change at all). We can calculate all the resulting position errors of these FK solutions from the desired foot position using simple vector subtraction (i.e. Position error vector = Target position vector subtract the current position vector of the foot). A satisfactory IK solution is found when the *position error magnitude* for a combination of joint variable changes is less than or equal to the chosen *position error tolerance*.

There are several ways to implement BASIK to search for a suitable IK solution using software, but only two different approaches will be described here, namely:

Approach 1: Perform a search for the combination of joint variable changes that produces the smallest error magnitude (after all position errors are calculated). If the tolerance is not met, change search parameters Δθ or Δ*d* or T and repeat.

Approach 2: Test if the error tolerance is satisfied immediately after an error magnitude is calculated, and exit the search early if the tolerance is met. If the tolerance is not met, change search parameters $\Delta\theta$ or Δd or T and repeat.

If an acceptable IK solution cannot be found in the first scan of the search space (Fig. 13), $\Delta\theta$ or Δd may be reduced. As shown in Fig. 12, the target position vector for the foot, ${}^{B}\boldsymbol{t}_{T}$, starts from the base frame origin B and points to the target frame origin T. The foot position vector ${}^{B}\boldsymbol{p}_{F}$ starts from the base frame B and points to the foot F.

$$ {}^{F}\boldsymbol{e}_{T} = {}^{B}\boldsymbol{t}_{T} - {}^{B}\boldsymbol{p}_{F} \quad \text{(or test error } {}^{F}\boldsymbol{e}_{Tt} = {}^{B}\boldsymbol{t}_{T} - {}^{B}\boldsymbol{p}_{Ft} \text{)} \tag{37} $$

The starting *position error vector* starts from the foot F and points to the target frame origin T. The goal is to find a suitable set of joint angle changes to get the Foot position F to end up at the Target position T, within an acceptable error range (or *tolerance*). We need to find the combination of joint angle (or displacement) changes that achieves an error vector magnitude $|{}^{F}\boldsymbol{e}_{T}|$ less than an acceptable error tolerance (e.g. 0.1 mm) within a reasonably short time, for quick real-time performance. 'Approach 1' requires calculating the FK solution for each and every possible combination of joint angle changes and then selecting the combination of changes that produces the smallest position error from the target position. This process repeats (if necessary) until the required error tolerance is satisfied.

The following algorithm demonstrates 'Approach 1':

1. A new target end-effector (foot) position T is issued (as an x, y, z coordinate relative to base frame B) and this must be quite close to the current position of the end-effector frame (but within the known workspace boundary). See Fig. 12. T should be very close to F.
2. All the FK solutions and test errors for each of the displacement combinations (as shown in Table 2) are calculated. ('Approach 1' stores these in an array)
3. A *minimum value search* is performed to find out which displacement combination produces the smallest error magnitude (smallest distance to target T) and this combination is marked as the current '*best IK solution*'.
4. If the smallest error found satisfies the acceptable *error tolerance*, then the IK solution is found so return to Step 1 for the next target, otherwise, the search parameters can be changed (e.g. Method '4a': reduce $\Delta\theta$ or Δd for a finer search, or Method '4b': the new target position/orientation T is set much closer to the current position/orientation F of the end-effector) and the process is repeated from Step 2 until the best or smallest error magnitude (for a set of joint angle changes or small displacements) satisfies the error tolerance criteria.

Another viable alternative to Methods '4a' and '4b' is Method '4c': 'If the smallest error found satisfies the acceptable error tolerance for a match, then the IK solution is found, otherwise, the most recent 'best IK solution' for this smallest error is used as the starting or current test configuration for ${}^{B}\boldsymbol{p}_{Ft}$, then return to Step 2 and keep repeating this process until the smallest error magnitude (for a set of joint angle changes or small displacements) satisfies the error tolerance criteria for an acceptable IK solution'. While searching for an IK solution, Method '4c' reduces the difference between the manipulator's end-effector frame and the target frame. Basically, 'feedback control' programming principles are used to find the BASIK solution using only the FK equations. In essence, Method '4c' allows the algorithm to search for small joint angle / displacement changes so that the current test joint angles / displacements converge towards an acceptable IK solution (that satisfies the error tolerance).

'Approach 2' is much faster and more efficient than 'Approach 1' because there may be no need to calculate the FK solution and error magnitude for every single combination of joint angle changes. Also, Step 3 in the previous algorithm can be avoided, but Step 2 will need to perform an additional comparison check to see if the position error tolerance is satisfied ('Method 4(c)' is not used here).

There is the possibility that an IK solution cannot be found if the $\Delta\theta$ or Δd values are too large, or if the target position T is too far away from the current foot position F (see Fig. 12), therefore, it is recommended that small displacement values and/or small incremental moves are chosen for the end-effector (or Foot F) to guarantee a stable and reliable IK solution every time. However, such displacements or incremental moves should be large enough to achieve reasonably fast calculation performance or an acceptable IK solution rate that can keep up with all target position updates. i.e. In general, the smaller the values of the error magnitudes, $|{}^{F}\boldsymbol{e}_{T}|$, the more stable and reliable the BASIK method will be, but the IK solution may end up too slow if a solution is not found in the first 'scan' of the search space. The search parameters and IK error tolerances can be set proportional to the range of motion for each link. The larger the *acceptable position error* or *tolerance* is for an IK solution, the faster a BASIK solution can be found.

B. Controlling position and orientation of the end-effector

The discussion in the previous section dealt mainly with finding an IK solution for a serial-link manipulator given a target position (origin of target frame T) which is very close to the origin of the end-effector frame E. We have so far only described how to move the origin of end-effector frame E to the origin of target frame T. We will now consider an extension to this incremental IK method and attempt to orient the $\underline{x}_{E}$ and $\underline{y}_{E}$ basis axes of end-effector frame E so that they point in the same directions as the corresponding $\underline{x}_{T}$ and $\underline{y}_{T}$ basis axes of the target frame T respectively. Note that only two of the three corresponding basis axes need alignment as long as both frames are 'right handed'. The complete incremental IK solution can be found for any type of serial-link arm. We must find the robot's joint variables to move the origin of frame E to the origin of frame T (which is only a short distance away) while the corresponding x and y (and consequently z) basis axes of both frames are made to point in the same directions, respectively, within an acceptable angular or error tolerance (i.e. $\underline{x}_{E}$ becomes colinear with $\underline{x}_{T}$ and $\underline{y}_{E}$ becomes colinear with $\underline{y}_{T}$). Consider the end-effector frame E for any type of multi-degree-of-freedom robot arm manipulator, as shown in Fig. 14.

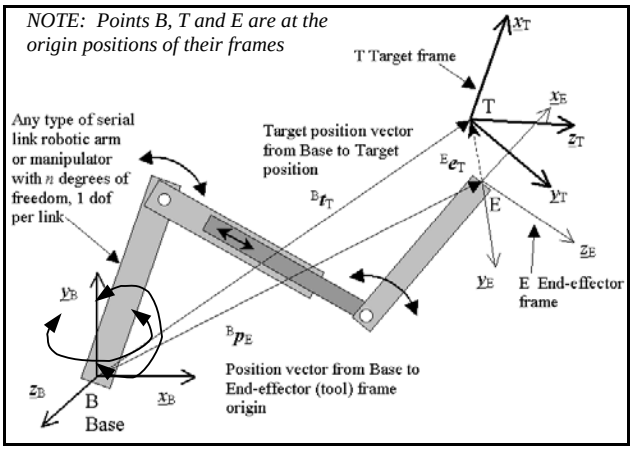

Figure 14. Base (B), End-effector (E) and Target (T) frames for any robot

The robot arm shown in Fig. 14 is just for illustrative purposes only and this discussion applies to *any* serial-link robot arm design with one degree of freedom per link. A link with a spherical joint (or 'ball joint') can be treated like two rotary links, where the first link has zero link length ($l = 0$), so each link has one rotation / displacement about one axis. The human upper arm (or humerus bone) is attached to the shoulder (spherical joint) and has 2 rotary degrees of freedom. It can be considered as 2 links, namely, one link with a shoulder-to-elbow link length attached to an 'invisible link' with zero link length but each link, including the zero-length link, has one rotary degree of freedom.

The Target frame can be specified relative to the Base frame of the robot using a standard 4x4 transformation matrix ${}^{B}\mathbf{T}_{T}$. The $\underline{x}_T$, $\underline{y}_T$ and $\underline{z}_T$ unit vectors representing the basis axes of frame T relative to the base frame B are obtained from the first 3 columns of the ${}^{B}\mathbf{T}_{T}$ 'target' matrix.

$$ {}^{B}\mathbf{T}_{T} = \begin{bmatrix} a_T & d_T & g_T & p_T \\ b_T & e_T & h_T & q_T \\ c_T & f_T & i_T & r_T \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (38) $$

$$ = \begin{bmatrix} \underline{x}_T & \underline{y}_T & \underline{z}_T & {}^{B}\boldsymbol{t}_T \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

where the direction vectors of the basis axes of frame T with respect to frame B are given by

$$ \underline{x}_T = a_T\, \underline{x}_B + b_T\, \underline{y}_B + c_T\, \underline{z}_B $$
$$ \underline{y}_T = d_T\, \underline{x}_B + e_T\, \underline{y}_B + f_T\, \underline{z}_B $$
$$ \underline{z}_T = g_T\, \underline{x}_B + h_T\, \underline{y}_B + i_T\, \underline{z}_B $$

The point vector of the frame T origin point relative to the frame B origin is

$$ {}^{B}\boldsymbol{t}_T = p_T\, \underline{x}_B + q_T\, \underline{y}_B + r_T\, \underline{z}_B $$

Similarly, the $\underline{x}_E$, $\underline{y}_E$ and $\underline{z}_E$ unit vectors representing the basis axes of frame E relative to the base frame B are obtained from the first 3 columns of the ${}^{B}\mathbf{T}_{E}$ matrix which is the FK transformation matrix of the entire manipulator, similar to the type found in (7), obtained by combining all the A-matrices for the manipulator. The manipulator transform for an n-link manipulator is thus

$$ {}^{B}\mathbf{T}_{E} = {}^{0}\mathrm{A}_{1}\ {}^{1}\mathrm{A}_{2} \ldots {}^{n-1}\mathrm{A}_{n} $$

(for any serial-link manipulator with n links ≥ 3)

$$ {}^{B}\mathbf{T}_{E} = \begin{bmatrix} a_E & d_E & g_E & p_E \\ b_E & e_E & h_E & q_E \\ c_E & f_E & i_E & r_E \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (39) $$

$$ = \begin{bmatrix} \underline{x}_E & \underline{y}_E & \underline{z}_E & {}^{B}\boldsymbol{p}_E \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

where the direction vectors of the basis axes of frame E and the position of point E with respect to frame B, are given by

$$ \underline{x}_E = a_E\, \underline{x}_B + b_E\, \underline{y}_B + c_E\, \underline{z}_B $$
$$ \underline{y}_E = d_E\, \underline{x}_B + e_E\, \underline{y}_B + f_E\, \underline{z}_B $$
$$ \underline{z}_E = g_E\, \underline{x}_B + h_E\, \underline{y}_B + i_E\, \underline{z}_B $$
$$ {}^{B}\boldsymbol{p}_E = p_E\, \underline{x}_B + q_E\, \underline{y}_B + r_E\, \underline{z}_B $$

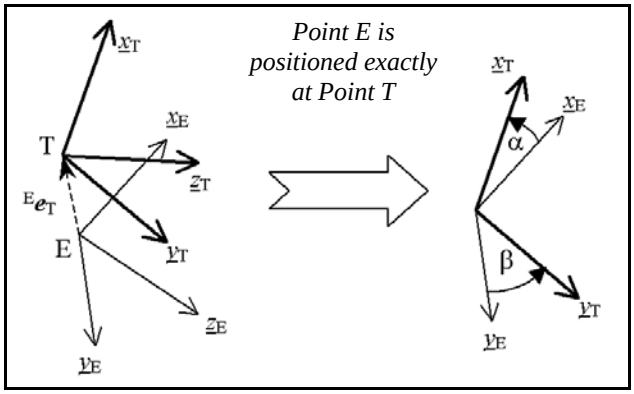

Figure 15. Superimposing the origins of both E and T frames to measure angles α and β (angles between the x and y axes, respectively)

Fig. 15 shows the error vector and angular differences between the x and y basis axes of frames E and T, the magnitudes of which all need to be 'driven towards 0' or reduced below an acceptable error tolerance value in order to obtain an acceptable IK solution. To achieve a suitable IK solution, the magnitude of the error vector $|{}^{E}\boldsymbol{e}_T|$ and the angles between pairs of corresponding $\underline{x}_E$, $\underline{x}_T$ and $\underline{y}_E$, $\underline{y}_T$ basis axes (i.e. α and β respectively) can be calculated and then combined into a 'total error' value $|e_{total}|$ which can be used to search for the best combination of joint variable changes. We may use the 'Scalar' or 'Dot Product'

operation on basis vectors $\underline{x}_E$ and $\underline{x}_T$ to find the angle between them, α. Likewise, we can perform the same operation on vectors $\underline{y}_E$ and $\underline{y}_T$ to find β. There is no need to do this for the 3rd pair of axes, vectors $\underline{z}_E$ and $\underline{z}_T$, because if the other two axes line up, the z axes will automatically be aligned relative to the x-y planes because both frames are 'right handed'. The solutions for both α and β can each range anywhere from 0° to 180° . Also, the magnitude of any basis (unit) vector is 1 so $|\underline{x}_E| = |\underline{x}_T| = |\underline{y}_E| = |\underline{y}_T| = 1$. The inner or 'Dot Product' operations can now be used to find α and β, or angular errors between the x and y axes.

$$\underline{x}_E \bullet \underline{x}_T = |\underline{x}_E|\,|\underline{x}_T| \cos\alpha = \cos\alpha \quad (40)$$
$$\underline{y}_E \bullet \underline{y}_T = |\underline{y}_E|\,|\underline{y}_T| \cos\beta = \cos\beta \quad (41)$$

It is useful to note that if α or β lies between 90° and 180°, the cosine function will return a negative value. The 'worst case' alignment between any two basis axes vectors is 180°, which gives cos(180°) = -1. If α or β are between 0° and 90°, the cosine function will return a positive value. The 'best case' for alignment between any two basis axis vectors is 0°, which gives cos(0°) = +1. Hence, *angular alignment error* between the $\underline{x}_E$ and $\underline{x}_T$ axes can be measured using a positive value like $1 - \cos(\alpha)$. If the angle $\alpha = 0$, cos (0°) = 1, so the alignment error is 1 - 1 = 0 (meaning zero alignment error). If α = 180°, cos(180°) = -1 so 1-(-1) = +2 which gives the largest or maximum value for alignment error. 'Angular alignment error' is an artificial term that ranges from 0 (perfect alignment) to +2 (worst case alignment) and is simply used as a measure of how poorly a pair of basis axes line up. We can designate e_{ax} as the angular alignment error between $\underline{x}_E$ and $\underline{x}_T$, and e_{ay} as the angular alignment error between $\underline{y}_E$ and $\underline{y}_T$.

$$e_{ax} = 1 - \cos\alpha = 1 - \underline{x}_E \bullet \underline{x}_T$$
$$= 1 - (a_E\, a_T + b_E\, b_T + c_E\, c_T) \quad (42)$$

$$e_{ay} = 1 - \cos\beta = 1 - \underline{y}_E \bullet \underline{y}_T$$
$$= 1 - (d_E\, d_T + e_E\, e_T + f_E\, f_T) \quad (43)$$

These values are useful for calculating an overall total error e_{total} which measures position error (or distance between the origin points of frame T and E) and axis alignment errors. An expression for e_{total} can be created to combine the position error and angular alignment error terms so that the best combination of joint variable changes can be found to minimize this 'total error'. 'Total error' can now be formulated using two 'weighting factors' which can be adjusted to scale the importance of each source of error. K_p is the factor which adjusts the contribution of the position error vector magnitude (or incremental step size to the next target position) $|^E\boldsymbol{e}_T|$ towards the 'total error'. K_a is the factor which adjusts the contribution of both e_{ax} and e_{ay} angular misalignment (error) values. These weighting factors are like 'gains' for a PID algorithm but they must always remain positive, i.e. e_{total} , e_{ax} and e_{ay} are always positive. We will call e_{pos} the error term due to position error (distance between E and T) and e_{ang} will be the error term due to the sum of angular alignment error values.

$$e_{pos} = K_p\, |^E\boldsymbol{e}_T| = K_p\, |(^B\boldsymbol{t}_T - {}^B\boldsymbol{p}_E)| \quad (> 0 \text{ positive}) \quad (44)$$
$$e_{ang} = K_a\,(e_{ax} + e_{ay}) \quad (> 0 \text{ positive}) \quad (45)$$
$$e_{total} = e_{pos} + e_{ang}$$
$$= K_p\, |^E\boldsymbol{e}_T| + K_a\,(e_{ax} + e_{ay}) \quad (> 0) \quad (46)$$

The values for K_p and K_a need to be adjusted so that a fair balance can be obtained between the contribution of position error and the contribution of angular alignment errors. The worst-case value for e_{pos} may be set equal to the worst-case value of e_{ang} if position error is just as important as alignment error for the axes. These weighting factors can be determined by monitoring the magnitudes of such position and angular alignment errors using 3D simulations. Accuracy and speed of the BASIK method depends largely on the value of the *total error tolerance* e_{total} chosen for an acceptable IK solution. Higher precision solutions usually require more iterations. The variable e_{total} can be used just like a 'test' error $|^F\boldsymbol{e}_{Tt}|$ in Table 2 which can be found for each possible combination of joint variable changes.

The same methods used in the previous section may be used to search for the best combination of joint variable changes to satisfy the tolerance for an acceptable IK solution. The smallest 'total error' value can be found in each scan of the search space using a minimum value search and this can be compared to the best 'total error' found so far. The BASIK method keeps searching for a suitable combination of joint displacement changes which produces an e_{total} value below a satisfactory *total error tolerance* value, e_{tol} . Hence, an acceptable IK solution is found only when $e_{total} \leq e_{tol}$. The error tolerance for the IK solution must be set by the programmer along with carefully selected values for search parameters like initial position displacement or step $|^E\boldsymbol{e}_T|$ and $\Delta\theta$ (or Δd for translational joints). This process can be automated by making such values proportional to link length, maximum range of motion or stroke length (for translation).

VII. Control and Simulation Software for BASIK

A model of a serial-link robotic manipulator or arm can be constructed by joining together different kinds of robot links chosen from a set of standard links (see Fig. 16). The geometric or D-H properties of each link can be modified or set based on real-world link geometry and dimensions taken from CAD drawings, like the kind shown in Table 1. Each link has only one rotary or sliding joint. A spherical joint can be treated as two rotary links in series, each with one rotary degree of freedom that rotates about an axis perpendicular to the axis of rotation of the other link, however, the first link has zero link length, so it appears as though the second link in the chain has two rotary degrees of freedom (e.g. Pitch and Yaw rotations) about one point. Each modeled link can be joined or connected to the end of an existing link, graphically, using software (see Fig. 17).

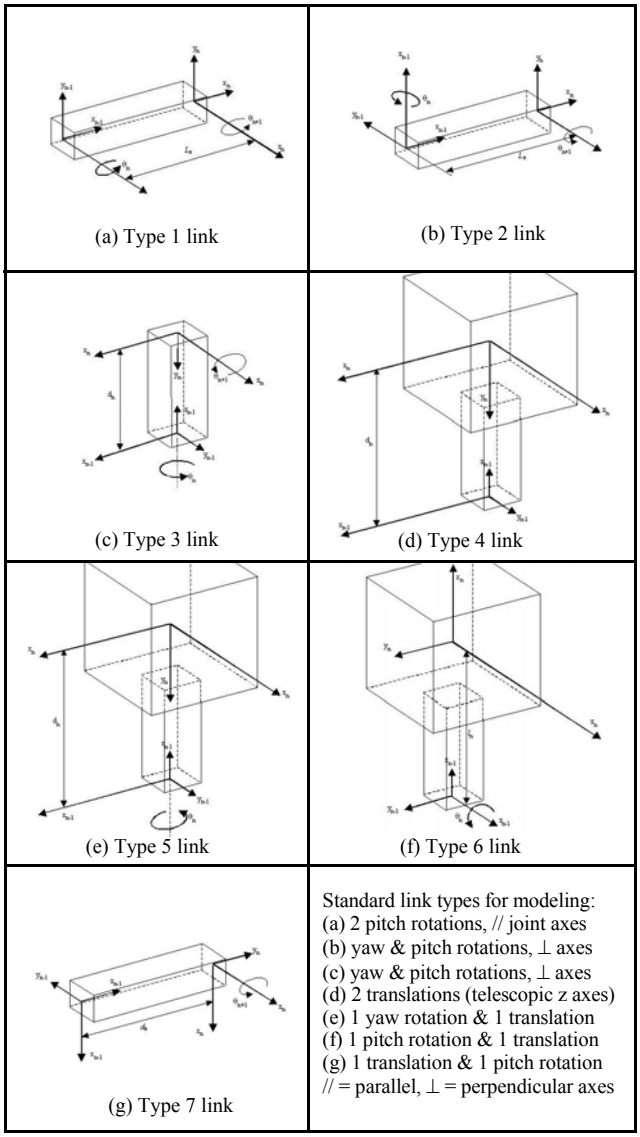

Figure 16. Standard robot arm link types (based on McKerrow [7])

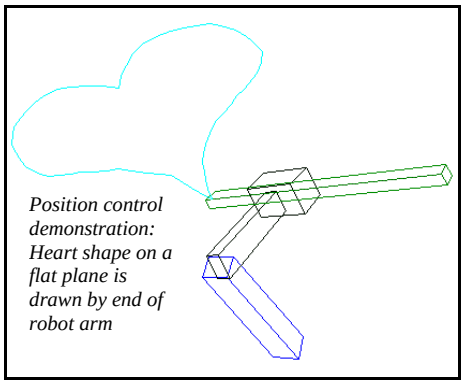

Figure 17. Robot model drawing heart shape on a flat plane using BASIK

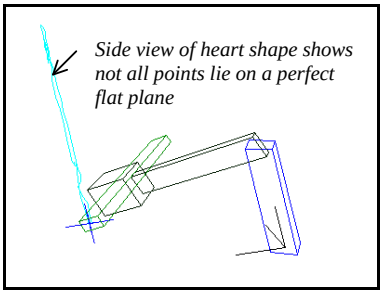

Figure 18. Side view of heart shape showing actual end-effector positions

The custom-designed robot model shown in Figs. 17 and 18 consists of three different links joined together, namely, a Type 1 (at the base), a Type 6, and a Type 4 link (at the end). Several coordinate points for drawing a 'heart shape' on a flat plane were set as the goal points for the 'end-effector' of this robot model. Fig. 17 shows the 'point-to-point' movements of the end-effector of the robot model. Fig. 18 shows the side view of this 'heart shape', showing the small errors in the positioning of the points (which should all lie on the same plane), due to large error tolerances used for the IK solution. Performance of this BASIK simulation software is typically smooth and fast on modern PCs built after 1996.

VIII. CONCLUSIONS

The BASIK method was described and has proven to be a simple, reliable 'general-purpose' method for achieving an accurate and robust IK solution for any kind of serial-link robot manipulator or arm design, regardless of the number or types of links used. Additional software is needed to control the desired 'shape' or 'configuration' of a robot arm having several redundant links. BASIK overcomes the serious disadvantages and limitations of popular IK methods. It can be easily adapted for use in general-purpose simulation and control programs to control all kinds of robot manipulators.

REFERENCES

[1] Cubero S., Force, Compliance and Position Control for a Pneumatic Quadruped Robot, Ph.D Thesis, Faculty of Engineering, 1998, USQ
[2] Hamlin G. J., Sanderson A. C., Tetrobot: A modular approach to parallel robotics, 1997, IEEE Robotics and Automation Magazine, 4(1), pp. 42-50
[3] Hamlin G., Sanderson A., TETROBOT: A Modular Approach to Reconfigurable Parallel Robotics, 1997, Kluwer Academic Publishing
[4] Naccarato F., Hughes P., Inverse Kinematics of Variable Geometry Truss Manipulators, 2007, Journal of Robotic Systems, 2(8), pp. 249-266, Wiley, DOI: 10.1002/rob.4620080207
[5] Cubero S., Billingsley J., A novel proportional gas valve for mechatronics applications, 1995, Proc 2nd Int Conf on Mechatronics and Machine Vision In Practice (M2VIP), ISBN: 962-442-0769
[6] Stephens R., Visual Basic Graphics Programming, 1997, Wiley, ISBN-10: 0471155330, ISBN-13: 978-0471155331
[7] McKerrow, P. J., Introduction to Robotics. (Chapters 3 and 4) Addison-Wesley, 1991, ISBN: 0-201-18240-8
[8] Paul R. P., Robot manipulators – Mathematics, programming and control. MIT, USA, 1991, ISBN: 0-262-16082-X
[9] Ranky P. G., Ho C. Y., Robot modelling: control and applications with software. IFS Publications Ltd. UK, 1985, ISBN: 0-903608-72-3, and Springer-Verlag, ISBN: 3-540-15373-X
[10] Fu K. S., Gonzalez R. C., Lee C. S. G., Robotics: Control, Sensing, Vision and Intelligence. McGraw-Hill, 1987, ISBN: 0-07-100421-1