

# IKBT: solving closed-form Inverse Kinematics with Behavior Tree

Dianmu Zhang, *Student Member, IEEE*, and Blake Hannaford, *Fellow, IEEE*

**Abstract**—Serial robot arms have complicated kinematic equations which must be solved to write effective arm planning and control software (the Inverse Kinematics Problem). Existing software packages for inverse kinematics often rely on numerical methods which have significant shortcomings. Here we report a new symbolic inverse kinematics solver which overcomes the limitations of numerical methods, and the shortcomings of previous symbolic software packages. We integrate Behavior Trees, an execution planning framework previously used for controlling intelligent robot behavior, to organize the equation solving process, and a modular architecture for each solution technique. The system successfully solved, generated a LaTeX report, and generated a Python code template for 18 out of 19 example robots of 4-6 DOF. The system is readily extensible, maintainable, and multi-platform with few dependencies. The complete package is available with a Modified BSD license on Github.

**Index Terms**—AI Reasoning Methods, Behavior Tree, Kinematics



## 1 INTRODUCTION

SYMBOLIC inverse kinematics analysis is a non-trivial task critical for operation and design of robot manipulators. Considering here serial non-redundant chains of up to six degrees of freedom, the inverse kinematics computation takes the desired end-effector pose as input (typically as a homogeneous transform), and solves for joint angles or joint displacements from the forward kinematic equations.

While there are many existing packages for numerical inverse kinematics [1] [2], these share common limitations including, finding only one of the multiple solutions, requirement of a starting value, dependence on the starting value, and problems with convergence near singular configurations.

Several groups have attempted to automate symbolic inverse kinematics analysis starting in the 1990's [3] [4], which laid the foundation for our work. Their work is reviewed and compared with ours in detail in the Discussion section.

In this work we develop an automated symbolic inverse kinematics package with the following goals:

- Create a highly extensible and modifiable architecture using a flexible behavior tree for solution logic and a modular design.
- Provide convenience features such as automatic documentation and code generation.
- Implementation in a modern open-source, cross-platform, programming language (Python)
- Require minimal dependencies outside of the standard Python distribution (mainly the symbolic manipulation package `sympy`).

Unlike others' work that mostly used a linear and inflexible method, we adapt Behavior Tree - popular in video game AI - to construct an expert system, "IKBT", that has the

logical reasoning power to solve inverse kinematics symbolically without human supervision. A Behavior Tree uses a directed graph to model intelligent agent behavior [5], [6], [7], [8]. Behavior Trees have the advantages of composability and scalability compared to finite state machines.

The main contributions of this work are:

- We compactly encode the inverse kinematics logic and strategy in a Behavior Tree (see Work Flow and Architecture section).
- We code each knowledge-based solver into a modular leaf, forming a "tool box" which is organized by the Behavior Tree (see Transformations and Solvers section).
- IKBT generates a dependency *graph* of joint variables in the solutions, includes all possible poses. Tracking these dependencies facilitates grouping variables into distinct solutions, essential to downstream control softwares for robots (see Solution Graph section).
- IKBT successfully solves complicated robots, such as the 6-DOF commercial robot manipulator PUMA 560 and successfully solved 18 out of 19 test robots (95% success rate) (see Results section).
- On average, IKBT generates symbolic solutions and source code in a few minutes on a normal PC. The same work load often takes a human expert hours to complete.
- IKBT generates a report of its results in  $\text{\LaTeX}$ , and generates code in Python and C++ creating functions to implement the derived solutions with domain (reachability) checking of numerical inputs (see Pose Validation section).
- Inverse kinematics solutions from IKBT are verifiable with numerical values (see Result Verification subsection under Results).

---

• D. Zhang and B. Hannaford are with the Department of Electrical Engineering, University of Washington, Seattle, WA, 98052.  
E-mail: blake@uw.edu

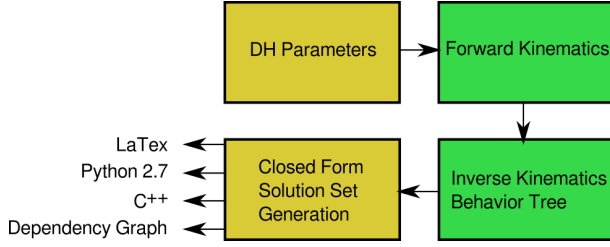


Fig. 1. **Work Flow.** Forward kinematics module computes symbolic kinematic equations to be solved ( $T_d = T_s$ ) given the input DH parameters. Then the equations are evaluated for closed-form inverse kinematics solutions to each joint variable. Upon solving a robot, along with the solutions, a dependency graph, Latex report, and Python/C++ code are generated as convenience feature.

## 2 WORK FLOW AND ARCHITECTURE

### 2.1 Work Flow

As shown in Fig. 1, the system input takes symbolic Denavit–Hartenberg (DH) parameters and calculates symbolic forward kinematic equations in the form of a  $4 \times 4$  homogeneous transformation. For a 6-DOF robot, the transformation matrix  $T_0^6$  is computed as

$$T_s = T_0^6 = T_0^1 T_1^2 T_2^3 T_3^4 T_4^5 T_5^6 \quad (1)$$

By convention, each transformation matrix takes a coordinate from the subscript frame and transforms it to the superscript frame ( $T_0^6$  transforms from frame 0 to frame 6).

We denote the desired robot end effector pose as  $T_d$ . Then the inverse kinematics problem can be stated as solving

$$T_d = T_0^6(q_1, \dots, q_6) \quad (2)$$

(where  $q_i$  are the unknown joint variables) for all sets of joint variables ( $q_i = \theta_i$  or  $d_i$ ) which satisfy 2. Related equations which can be used to find solvable equations include:

$$[T_0^1]^{-1} T_d = [T_0^1]^{-1} T_s \quad (3)$$

$$[T_1^2]^{-1} [T_0^1]^{-1} T_d = [T_1^2]^{-1} [T_0^1]^{-1} T_s \quad (4)$$

$$[T_{n-2}^{n-1}]^{-1} \dots [T_0^1]^{-1} T_d = [T_4^5]^{-1} \dots [T_0^1]^{-1} T_s \quad (5)$$

IKBT first symbolically calculates and simplifies these intermediate results (3 - 5) to augment (1). Each of these matrix equations creates 12 scalar equations (one for each element in the first three rows) which can be searched for solvable equations. After generating these matrices, each scalar equation is categorized by the number of unsolved joint variables,  $q_i$ , into three lists according to the number of unsolved variables in each equation (1, 2, and 3-or-more unknowns). As each variable is solved, this scan is repeated. The current toolbox examines equations containing 1- and 2-unsolved variables.

The Behavior Tree’s leaf nodes transform, or identify and solve, a particular kind of equation (See detail in the following section “Transforms and Solvers”). For example, one pair of nodes identifies and solves scalar equations of one-unknown of the form  $A = \sin(B\theta_j + C)$  or  $A = \cos(B\theta_j + C)$  where  $A, B, C$  are known expressions, and  $\theta_j$  is unknown.

After all joint variables are solved, the solution graph and the solution vectors ( $2^n$  joint vectors in symbolic form correctly associating the multiplicity of each variable) are constructed. A  $\LaTeX$  report, C++ and Python code, containing symbolic solutions for all possible poses, is also generated.

### 2.2 Architecture

Behavior Trees have been explored in the context of humanoid robot control [6], [9], [10], collaborative robotics [8], [11], and as a modeling language for intelligent robotic surgical procedures [12], [13].

The work reported here is the first to our knowledge to use Behavior Trees to encode algorithms for reasoning about and solving mathematical equations symbolically. When implementing intelligent behavior with Behavior Trees, the designer of a robotic control system breaks the task down into modules (Behavior Tree leaves) which return either “success” or “failure” when called by parent nodes. Higher level nodes define composition rules to combine the leaves including: Sequence, Selector, and Parallel node types which also return “success” or “failure”. A Sequence node defines the order of execution of leaves and returns success if all leaves succeed in order. A Selector node (called “Priority” by some authors) tries leaf behaviors in a fixed order, returns success when a node succeeds, and returns failure if all leaves fail. We also implemented a “Parallel” node (represented as “OR” in Fig. 2), which executes all leaves regardless of their return status, and returns success if any one of the leaves succeeds. The IKBT structure used for our current results is shown in Fig.2.

Before solving, IKBT looks through 1- and 2- unknown equation lists, and applies sum-of-angle and substitution transformations which may reduce number of unknown variables. The “Assigner” node assigns the current variable to all solvers. For each joint variable, it tries out all solvers in the toolbox until it is solved (or we reach the maximum trial number). When a joint variable is solved, the solver marks it as solved, and reduces the number of unsolved variables by one, for all equations involved this variable. When multiple solvers can solve a joint variable, the solutions are compared using a “Ranker” node which selects preferred solution forms over others. For example,  $\theta_4 = \text{atan2}(y, x)$  is preferred over  $\theta_4 = \arcsin(y/r)$  because it has only one solution. IKBT repeats this until all variables are solved. Finally the solutions, report, code generation, and dependency graph are generated.

## 3 TRANSFORMATIONS AND SOLVERS

In the following subsections,  $\theta_i$  and  $d_i$  represent rotatory and prismatic joint variables ( $q_i$ ).  $a, b, c$ , etc., stand for known constant DH parameters.

### 3.1 Transformations

Transformation nodes make equations easier to solve by reducing the number of unknown variables.

- 1) Sum of angle transform

$$\sin(\theta_x \pm \theta_y) \rightarrow \sin(\theta_{xy})$$

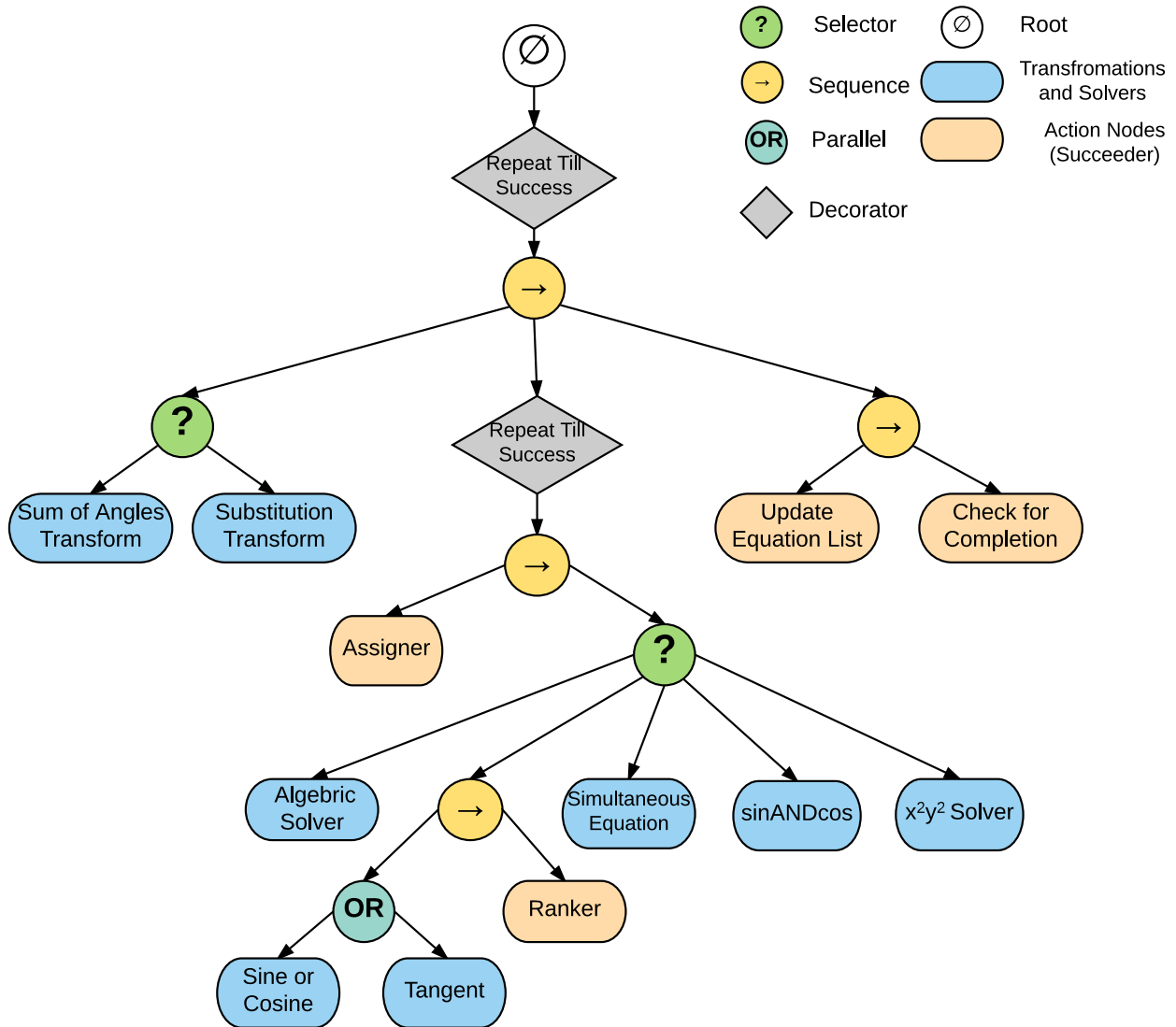


Fig. 2. **IKBT Structure.** Node type explanation: Action nodes (leaves) carry out specific tasks, and returns SUCCESS or FAILURE. Succeeder is a special type of action nodes that only returns SUCCESS. Selector node ticks its children in turn, returns SUCCESS and stops if one of the children succeeds, otherwise returns FAILURE. Sequence node only returns SUCCESS if all its children succeed. Parallel node tries out all its children regardless of their return status, returns SUCCESS if any child succeeds.

$$\cos(\theta_x \pm \theta_y) \rightarrow \cos(\theta_{xy})$$

Although the sum-of-angle simplification is done by sympy's `simplify` operation, creation of a new variable ( $\theta_{xy}$ ) is done by this node.

- 2) Substitution transform: looks for two equations such that one contains the other, and replaces the partial expression with a unknown value. For example, the following pair of equations:

$$\sin(\theta_x) + a \cdot \cos(\theta_y) = b$$

$$a \cdot \cos(\theta_y) = c$$

The first equation is transformed to:

$$\sin(\theta_x) + c = b$$

Eliminating one unknowns so that  $\theta_x$  can be solved.

### 3.2 Rule-based Solvers

The IKBT contains a set of solvers that identifies an expression that fits a rule set and return the respective solutions. These rules are used by human experts when solving inverse kinematics problems, and not are specific to any DOF or robot configuration.

- 1) algebraic solver  
Identifies pattern

$$a + b\theta = c$$

where  $b \neq 0$ . Solves for

$$\theta = \frac{c - a}{b}$$

as well as

$$a + b\theta_x = c$$

giving

$$d_x = \frac{c - a}{b}$$

- 2) sine or cosine solver  
Identifies pattern

$$\sin(\theta) = a, \quad \cos(\theta) = b$$

Solves for

$$\theta = \arcsin(a), \quad \theta = \arccos(b)$$

or

$$\theta = \pi - \arcsin(a), \quad \theta = -\arccos(b)$$

- 3) tangent solver  
identifies a pattern in two equations containing

$$\sin(\theta) = aC_1 \quad \text{and} \quad \cos(\theta) = bC_2$$

If neither  $C_1$  or  $C_2$  contain unsolved variables:

$$\theta = \text{atan2}\left(\frac{a}{C_1}, \frac{b}{C_2}\right)$$

Sometimes  $C_1$  and  $C_2$  contain common unsolved variables, which can be canceled out by division. In this case we use a new coefficient  $C$ :

$$C = \frac{C_1}{C_2}$$

$$\theta = \text{atan2}\left(\frac{a}{C}, b\right) \quad C > 0$$

$$\theta = \text{atan2}\left(-\frac{a}{C}, -b\right) \quad C < 0$$

Terms which are solvable by tangent solver are often also solvable by sine or cosine solver. As shown in Fig 2, IKBT takes this into consideration by comparing the solutions from the above-mentioned solvers, and determines the optimal solution. This selection is done by the Ranking node.

- 4) Sine and cosine solver  
Identifies

$$a \cdot \sin(\theta) + b \cdot \cos(\theta) = 0$$

giving

$$\theta = \text{atan2}(-b, a), \quad \theta = \text{atan2}(-b, a) + \pi$$

as well as

$$a \cdot \sin(\theta) + b \cdot \cos(\theta) = c$$

giving

$$\theta = \text{atan2}(a, b) + \text{atan2}(\pm\sqrt{a^2 + b^2 - c^2}, c)$$

- 5) Simultaneous equation solver [3] Identifies two equations:

$$a \sin(\theta) + b \cos(\theta) = c \quad a \cos(\theta) - b \sin(\theta) = d$$

Giving the solution:

$$\theta = \text{atan2}(ac - bd, ad + bc)$$

- 6)  $x^2y^2$  solver

Identifies two equations that contain  $P_x$ ,  $P_y$ , and/or  $P_z$ , that can be squared and added together to cancel out unsolved variables (other than the intended variable), and get a new equation with pattern [14]:

$$-\sin(\theta)P_x + \cos(\theta)P_y = d$$

Giving solutions:

$$\theta = \text{atan2}(P_y, P_x) - \text{atan2}(d, \pm\sqrt{P_x^2 + P_y^2 - d^2})$$

## 4 SOLUTION GRAPH

### 4.1 Origins of Dependency

The solutions produced by inverse kinematics are typically interdependent in that results obtained early in the process are used to solve later results. For example, one may have

$$\theta_4 = \text{asin}\left(\frac{1}{l_4}(Pz - l_3 + l_5 \cos(\theta_{45}))\right)$$

in which  $\theta_4$  depends on  $\cos(\theta_{45}) = \cos(\theta_4 + \theta_5)$  as well as some constants. In principle, it is possible to substitute these dependencies until there are no joint variables on the right hand side, but this makes the solutions difficult to compare with previously published hand solutions. In the above example, there are two solutions to the  $\text{asin}()$  operator, and additional multiplicity could come from the solution method for  $\theta_{45}$ .

Thus the two sources of multiple solutions are: A) each joint variable may have multiple solutions due to its solver's characteristic; and B) dependence of the solution on other solved joint variables. For example,

$$\cos(\theta_1) = a \quad (6)$$

$$\theta_1 = \begin{cases} \theta_{1s1} = \arccos(a) \\ \theta_{1s2} = -\arccos(a) \end{cases}$$

(where we have used the subscript  $s$  to separate joint numbers from solution numbers. For example,  $\theta_{1s2}$  means solution 2 of  $\theta_1$ ). Now that  $\theta_1$  is solved, we can write

$$\sin(\theta_2) + \sin(\theta_1) = b \quad (7)$$

$$\theta_2 = \begin{cases} \theta_{2s1} = \arcsin(b - \sin(\theta_{1s1})) \\ \theta_{2s2} = \pi - \arcsin(b - \sin(\theta_{1s1})) \\ \theta_{2s3} = \arcsin(b - \sin(\theta_{1s2})) \\ \theta_{2s4} = \pi - \arcsin(b - \sin(\theta_{1s2})) \end{cases}$$

In the resulting graph (shown in Fig. 3 a), each joint solution (e.g.  $\theta_{2s1}$ ,  $\theta_{2s2}$ , etc.) is a node. A parent node is the node that appears in another node's solution expressions, in this example,  $\theta_{1s1}$  is the parent of  $\theta_{2s2}$ . A node and its parent/child node are connected with an edge.

### 4.2 Redundancy Detection and Dependency Tracking

When building a dependency graph, we implemented redundancy elimination to ensure the correct relations between joint variables. Redundancy is defined as a dependency that traced back to a higher level parent can be mediated by a lower level and direct parent. If a joint variable  $\theta_6$  has the following solution:

$$\theta_6 = \text{atan2}(a + \cos(\theta_4), \sin(\theta_5)b)$$

And  $\theta_5$  has solution:

$$\theta_5 = \arccos(l + \cos(\theta_4))$$

Though the solution of  $\theta_6$  involves both  $\theta_4$  and  $\theta_5$ , its dependency to  $\theta_4$  is redundant. Given that  $\theta_5$  is also dependent on  $\theta_4$ , the effects of choosing different  $\theta_4$  values (if applicable) on  $\theta_6$  are conveyed through  $\theta_5$ . Therefore, when building a graph, only the edges between direct child-parents are added, in this case, Edge  $(\theta_6, \theta_5)$  and Edge  $(\theta_5, \theta_4)$ . Shown in Fig. 3 b).

Classical search algorithms (breadth-first search and depth-first search) are used to traverse the graph and find correct ancestor nodes, where the current variable is the start point and the ancestors are the goals.

### 4.3 Grouping Variables

As required by many planning and control softwares, IKBT is capable of grouping variables into solution sets that have all possible joint configurations for the given end-effector configuration. To generate correct sets of solutions, the following steps are carried out to match the variables: First, all parents nodes are extracted from each solution expression, and formed subsets of variables. Secondly, the subsets are sorted by size of their content. Search starts from the largest subsets, and looks for the variables that are a part of the joint space, but not in the set, till all variables are found. A scoring system is applied on all subsets (other than the starting set) to focus the search on the more likely candidate first.

Using the Fig. 3 a) as an example, the solutions can be grouped into:  $[\theta_{1s1}, \theta_{2s1}]$ ,  $[\theta_{1s1}, \theta_{2s2}]$ ,  $[\theta_{1s2}, \theta_{2s3}]$ , and  $[\theta_{1s2}, \theta_{2s4}]$ .

### 4.4 Graph Representation

The multiple dependencies can be linked by a common dependency further up, or they can be independent. Although traditionally this structure is represented as a tree, we discovered cases in which variables have multiple independent "parents" and thus a graph is required instead.

For example,  $\theta_1$  and  $\theta_2$  are independent to each other:

$$\begin{aligned}\theta_{1s1} &= \arcsin(a) \\ \theta_{1s2} &= -\arcsin(a) + \pi \\ \theta_{2s1} &= \arccos(b) \\ \theta_{2s2} &= -\arccos(b)\end{aligned}$$

And  $\theta_3$  depends on both  $\theta_1$  and  $\theta_2$ :

$$\begin{aligned}\theta_{3s1} &= \arccos(a + \cos(\theta_{1s1}) + \text{atan2}(b, \sin(\theta_{2s1}))c) \\ \theta_{3s2} &= \arccos(a + \cos(\theta_{1s1}) + \text{atan2}(b, \sin(\theta_{2s2}))c) \\ \theta_{3s3} &= \arccos(a + \cos(\theta_{1s2}) + \text{atan2}(b, \sin(\theta_{2s1}))c) \\ \theta_{3s4} &= \arccos(a + \cos(\theta_{1s2}) + \text{atan2}(b, \sin(\theta_{2s2}))c)\end{aligned}$$

The dependency graph is shown in Fig. 3 c).

One example robot solution in Results section shows the necessity of using graph representation.

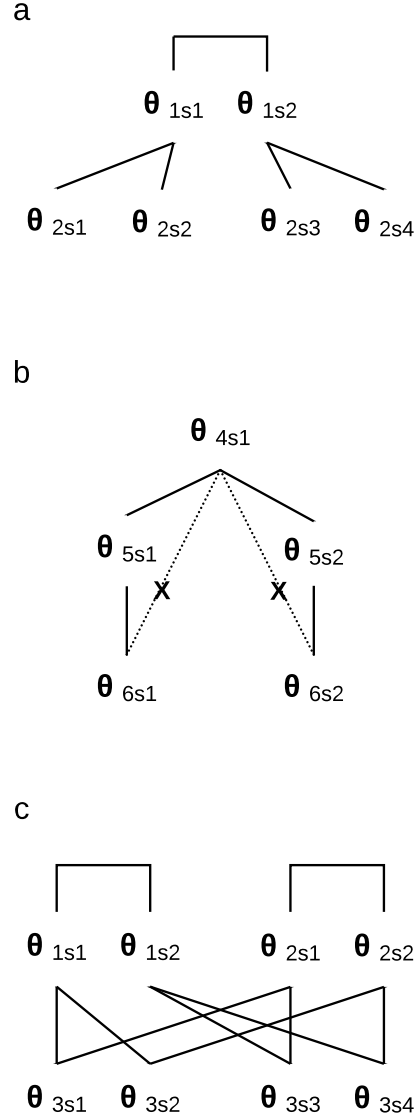


Fig. 3. **Sample Solution Graph.** a) Simple example solution graph explains the origins of multiplicity. b) Redundancy pruning, 'x' marks the dependent relations that are not included in the graph. c) Example of a case of variables with multiple independent parents.

## 5 VERIFICATION

### 5.1 Solution Verification

To prove that the inverse kinematics solution equations from IKBT are correct, we conducted the following verification process, as shown in Fig. 4 . First, we constructed a valid numerical transformation matrix from a reachable pose. Then used numbers from the transformation matrix and the inverse kinematics solutions equations to get the numerical values for each pose. If the inverse kinematics solutions are correct, one of the poses should match the starting pose value. Next, compute the forward kinematics using each numerical pose. If the resulted transformation matrix is the same (within a stringent range) as the starting matrix, then we can safely draw the conclusion that this pose has correct inverse kinematics solution.

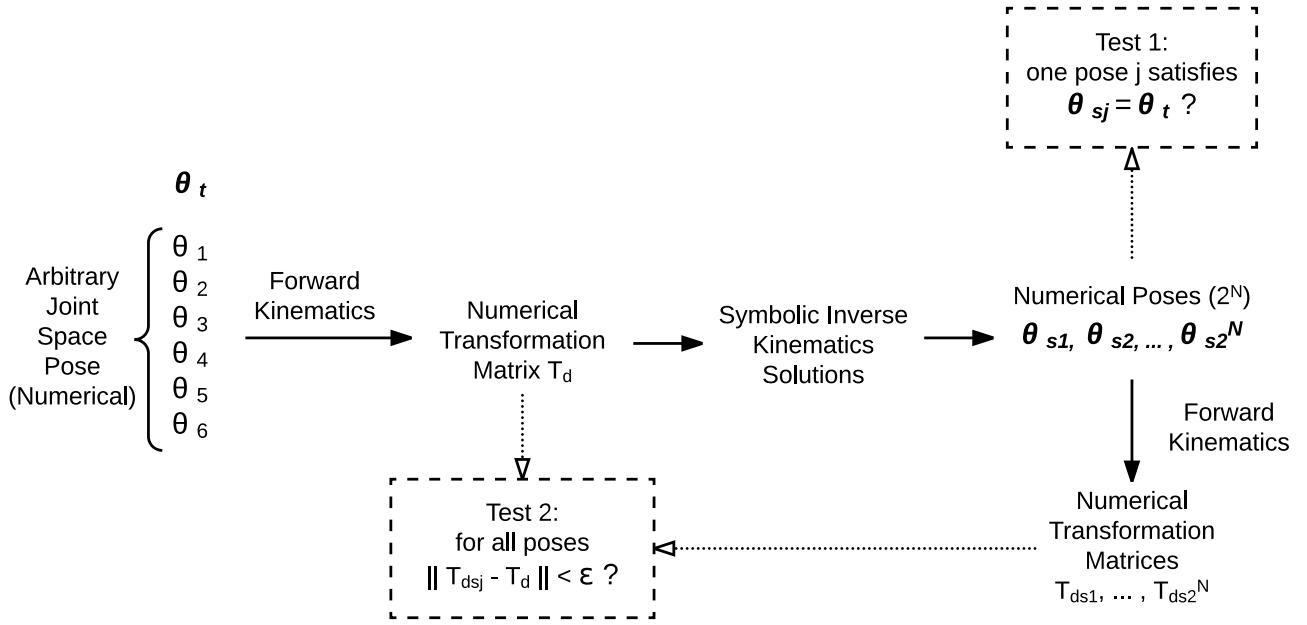


Fig. 4. **Result Verification.** A numerical 4x4 homogeneous transformation matrix,  $T_d$ , is constructed from a reachable pose. Numerical joint space poses are computed from  $T_d$  using the closed-form solutions. For each solution pose, forward kinematics is calculated. The resulting transform matrices are compared against the original matrix, matching value is indicative of correct IKBT inverse kinematics analysis.

## 5.2 Pose Validation

If a pose is not reachable by the robot (for example due to distance of a point extending beyond the length of the arm, but not considering joint limits), at least in generated code output, the solution must have a means to detect this case. In inverse kinematic solution equations, unreachable poses generate intermediate values outside the domain of transcendental functions, for example:

$$\theta_2 = \arcsin(x) \quad x = 1.2$$

or would require complex joint angles:

$$d_3 = \sqrt{x} \quad x = -5$$

Both the C++ and Python output modules of IKBT generate code which checks numerical arguments of inverse trig functions and square roots for such cases and returns a flag to indicate an unreachable pose.

## 6 RESULTS

### 6.1 General performance

We tested IKBT on many sets of DH parameters, representing serial arm robot designs (including commercial robots, and solved design examples from student homework), the successful solving rate is listed in Table 1. As the DOF number increases, the problem becomes more complex and the success rate decreases. In general it solves most of the robots, up to 6 DOF. Note that IKBT can solve robots regardless of their configurations, e.g. IKBT does not require robots having three intersecting axes.

Source code can be found at: <https://github.com/uw-biorobotics/IKBT>. The DH parameters of all these robots are stored as part of the

source code (in `ik_robots.py`), for purpose of testing and reproducing the results. Instructions are on the GitHub page.

### 6.2 Sample solutions PUMA 560

Here PUMA560 is used as an example to illustrate how IKBT solves inverse kinematics problems. PUMA 560 was a commercial robot with six rotary joints and four joint offsets, well-known for its challenging inverse kinematics properties. The PUMA 560 has three axes intersecting at its wrist. The known variables are:  $a_2$ ,  $a_3$ ,  $d_3$ , and  $d_4$ .

First forward kinematics was calculated from DH parameters:

$$T_d = T_s$$

$$T_0^6 = T_0^1 T_1^2 T_2^3 T_3^4 T_4^5 T_5^6$$

(where  $T_d$  is the “desired position”,  $T_s$  symbolic expressions,  $T_0^6$  transformation matrix from frame 0 to 6)

$$T_0^6 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{22} & r_{23} & P_y \\ r_{31} & r_{32} & r_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = [v1 \quad v2 \quad v3 \quad v4]$$

number of DOF	Test examples	Solved
4	4	4
5	10	10
6	5	4

TABLE 1  
IKBT test results

TABLE 2  
PUMA560 DH parameters

Link	$\alpha_{N-1}$	$a_{N-1}$	$d_N$	$\theta_N$
1	0	0	0	$\theta_1$
2	$-\pi/2$	0	0	$\theta_2$
3	0	$a_2$	$d_3$	$\theta_3$
4	$-\pi/2$	$a_3$	$d_4$	$\theta_4$
5	$\pi/2$	0	0	$\theta_5$
6	$-\pi/2$	0	0	$\theta_6$

$v_1 =$

$$\begin{bmatrix} c_6(-c_1 s_{23} s_5 + c_5(c_1 c_{23} c_4 + s_1 s_4)) - s_6(c_1 c_{23} s_4 - c_4 s_1) \\ c_6(c_5(-c_1 s_4 + c_{23} c_4 s_1) - s_1 s_{23} s_5) - s_6(c_1 c_4 + c_{23} s_1 s_4) \\ -c_6(c_{23} s_5 + c_4 c_5 s_{23}) + s_{23} s_4 s_6 \\ 0 \end{bmatrix}$$

$v_2 =$

$$\begin{bmatrix} -c_6(c_1 c_{23} s_4 - c_4 s_1) - s_6(-c_1 s_{23} s_5 + c_5(c_1 c_{23} c_4 + s_1 s_4)) \\ -c_6(c_1 c_4 + c_{23} s_1 s_4) - s_6(c_5(-c_1 s_4 + c_{23} c_4 s_1) - s_1 s_{23} s_5) \\ c_6 s_{23} s_4 + s_6(c_{23} s_5 + c_4 c_5 s_{23}) \\ 0 \end{bmatrix}$$

$$v_3 = \begin{bmatrix} -c_1 c_5 s_{23} - s_5(c_1 c_{23} c_4 + s_1 s_4) \\ -c_5 s_1 s_{23} - s_5(-c_1 s_4 + c_{23} c_4 s_1) \\ -c_{23} c_5 + c_4 s_{23} s_5 \\ 0 \end{bmatrix}$$

$$v_4 = \begin{bmatrix} a_2 c_1 c_2 + a_3 c_1 c_{23} - c_1 d_4 s_{23} - l_3 s_1 \\ a_2 c_2 s_1 + a_3 c_{23} s_1 + c_1 l_3 - d_4 s_1 s_{23} \\ -a_2 s_2 - a_3 s_{23} - c_{23} d_4 \\ 1 \end{bmatrix}$$

where  $c_1 = \cos(\theta_1)$ ,  $s_{23} = \sin(\theta_2 + \theta_3)$  etc.  
IKBT solved these variables in the following order:

1)  $\theta_1$ , chosen solver: sinANDcos

$$\begin{aligned} \theta_{1s1} &= \text{atan2}(Px, -Py) + \\ &\quad \text{atan2}(\sqrt{Px^2 + Py^2 - d_3^2}, -d_3) \\ \theta_{1s2} &= \text{atan2}(Px, -Py) + \\ &\quad \text{atan2}(-\sqrt{Px^2 + Py^2 - d_3^2}, -d_3) \end{aligned}$$

2)  $\theta_3$ , chosen solver:  $x^2 y^2$

$$\begin{aligned} \theta_{3s1} &= \text{atan2}(-2a_2 d_4, 2a_2 a_3) + \\ &\quad \text{atan2}(\sqrt{s - (t + (Px \cos(\theta_{1s1}) + Py \sin(\theta_{1s1}))^2)^2}, \\ &\quad t + (Px \cos(\theta_{1s1}) + Py \sin(\theta_{1s1}))^2) \\ \theta_{3s2} &= \text{atan2}(-2a_2 d_4, 2a_2 a_3) + \\ &\quad \text{atan2}(-\sqrt{s - (t + (Px \cos(\theta_{1s1}) + Py \sin(\theta_{1s1}))^2)^2}, \\ &\quad t + (Px \cos(\theta_{1s1}) + Py \sin(\theta_{1s1}))^2) \\ \theta_{3s3} &= \text{atan2}(-2a_2 d_4, 2a_2 a_3) + \\ &\quad \text{atan2}(\sqrt{s - (t + (Px \cos(\theta_{1s2}) + Py \sin(\theta_{1s2}))^2)^2}, \\ &\quad t + (Px \cos(\theta_{1s2}) + Py \sin(\theta_{1s2}))^2) \\ \theta_{3s4} &= \text{atan2}(-2a_2 d_4, 2a_2 a_3) + \\ &\quad \text{atan2}(-\sqrt{s - (t + (Px \cos(\theta_{1s2}) + Py \sin(\theta_{1s2}))^2)^2}, \\ &\quad t + (Px \cos(\theta_{1s2}) + Py \sin(\theta_{1s2}))^2) \end{aligned}$$

where,

$$\begin{aligned} s &= 4a_2^2 a_3^2 + 4a_2^2 d_4^2 \\ t &= Pz^2 - a_2^2 - a_3^2 - d_4^2 \end{aligned}$$

3)  $\theta_2$ , chosen solver: simultaneous equation

$$\begin{aligned} \theta_{23s1} &= \text{atan2}(Pz(-a_2 \cos(\theta_{3s3}) - a_3) - \\ &\quad (-Px \cos(\theta_{1s2}) - \\ &\quad Py \sin(\theta_{1s2}))(a_2 \sin(\theta_{3s3}) - d_4), \\ &\quad Pz(a_2 \sin(\theta_{3s3}) - d_4) + \\ &\quad (-Px \cos(\theta_{1s2}) - Py \sin(\theta_{1s2})) \\ &\quad (-a_2 \cos(\theta_{3s3}) - a_3)) \\ \theta_{23s2} &= \text{atan2}(Pz(-a_2 \cos(\theta_{3s2}) - a_3) - \\ &\quad (-Px \cos(\theta_{1s1}) - \\ &\quad Py \sin(\theta_{1s1}))(a_2 \sin(\theta_{3s2}) - d_4), \\ &\quad Pz(a_2 \sin(\theta_{3s2}) - d_4) + \\ &\quad (-Px \cos(\theta_{1s1}) - Py \sin(\theta_{1s1})) \\ &\quad (-a_2 \cos(\theta_{3s2}) - a_3)) \\ \theta_{23s3} &= \text{atan2}(Pz(-a_2 \cos(\theta_{3s1}) - a_3) - \\ &\quad (-Px \cos(\theta_{1s1}) - \\ &\quad Py \sin(\theta_{1s1}))(a_2 \sin(\theta_{3s1}) - d_4), \\ &\quad Pz(a_2 \sin(\theta_{3s1}) - d_4) + \\ &\quad (-Px \cos(\theta_{1s1}) - Py \sin(\theta_{1s1})) \\ &\quad (-a_2 \cos(\theta_{3s1}) - a_3)) \\ \theta_{23s4} &= \text{atan2}(Pz(-a_2 \cos(\theta_{3s4}) - a_3) - \\ &\quad (-Px \cos(\theta_{1s2}) - \\ &\quad Py \sin(\theta_{1s2}))(a_2 \sin(\theta_{3s4}) - d_4), \\ &\quad Pz(a_2 \sin(\theta_{3s4}) - d_4) + \\ &\quad (-Px \cos(\theta_{1s2}) - Py \sin(\theta_{1s2})) \\ &\quad (-a_2 \cos(\theta_{3s4}) - a_3)) \end{aligned}$$

4)  $\theta_2$ , chosen solver: algebraic solver

$$\begin{aligned} \theta_{2s1} &= \theta_{23s2} - \theta_{3s2} \\ \theta_{2s2} &= \theta_{23s4} - \theta_{3s4} \\ \theta_{2s3} &= \theta_{23s1} - \theta_{3s3} \\ \theta_{2s4} &= \theta_{23s3} - \theta_{3s1} \end{aligned}$$





$$\begin{aligned}\theta_{6s6} &= \text{atan2}\left(-\frac{1}{\sin(\theta_{5s3})}(-r_{12} \sin(\theta_{23s2}) \cos(\theta_{1s1}) - r_{22} \sin(\theta_{1s1}) \sin(\theta_{23s2}) - r_{32} \cos(\theta_{23s2})), \right. \\ &\quad \left. \frac{1}{\sin(\theta_{5s3})}(-r_{11} \sin(\theta_{23s2}) \cos(\theta_{1s1}) - r_{21} \sin(\theta_{1s1}) \sin(\theta_{23s2}) - r_{31} \cos(\theta_{23s2}))\right) \\ \theta_{6s7} &= \text{atan2}\left(-\frac{1}{\sin(\theta_{5s7})}(-r_{12} \sin(\theta_{23s3}) \cos(\theta_{1s1}) - r_{22} \sin(\theta_{1s1}) \sin(\theta_{23s3}) - r_{32} \cos(\theta_{23s3})), \right. \\ &\quad \left. \frac{1}{\sin(\theta_{5s7})}(-r_{11} \sin(\theta_{23s3}) \cos(\theta_{1s1}) - r_{21} \sin(\theta_{1s1}) \sin(\theta_{23s3}) - r_{31} \cos(\theta_{23s3}))\right) \\ \theta_{6s8} &= \text{atan2}\left(-\frac{1}{\sin(\theta_{5s5})}(-r_{12} \sin(\theta_{23s4}) \cos(\theta_{1s2}) - r_{22} \sin(\theta_{1s2}) \sin(\theta_{23s4}) - r_{32} \cos(\theta_{23s4})), \right. \\ &\quad \left. \frac{1}{\sin(\theta_{5s5})}(-r_{11} \sin(\theta_{23s4}) \cos(\theta_{1s2}) - r_{21} \sin(\theta_{1s2}) \sin(\theta_{23s4}) - r_{31} \cos(\theta_{23s4}))\right)\end{aligned}$$

$$\begin{bmatrix} -0.15720 & 0.97939 & 0.12682 & -1.68075 \\ -0.59374 & -0.19634 & 0.78033 & 1.33902 \\ 0.78915 & 0.04737 & 0.61237 & -4.83025 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We got the same values compared to the original T matrix for all solution poses, with differences  $\approx 10^{-5}$ . This result unequivocally proves that IKBT's symbolic inverse kinematics analysis is correct.

### 6.3 Example Solution - Robot without 3 intersecting axes

Previous software packages which perform inverse kinematics analysis usually require the robot to have three intersecting axes (such as the popular ROS package). To demonstrate IKBT's flexibility in handling robots with different configurations. We select the example of "Chair Helper", a 5 DOF robot without three intersecting axes (Table 4)<sup>1</sup>.

Forward kinematics:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & Px \\ r_{21} & r_{22} & r_{23} & Py \\ r_{31} & r_{32} & r_{33} & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix} = [v_1 \quad v_2 \quad v_3 \quad v_4]$$

$$v_1 = \begin{bmatrix} -c_2 s_3 s_5 + c_5 (c_2 c_3 c_4 + s_2 s_4) \\ c_5 (-c_2 s_4 + c_3 c_4 s_2) - s_2 s_3 s_5 \\ c_3 s_5 + c_4 c_5 s_3 \\ 0 \end{bmatrix}$$

$$v_2 = \begin{bmatrix} -c_2 c_5 s_3 - s_5 (c_2 c_3 c_4 + s_2 s_4) \\ -c_5 s_2 s_3 - s_5 (-c_2 s_4 + c_3 c_4 s_2) \\ c_3 c_5 - c_4 s_3 s_5 \\ 0 \end{bmatrix}$$

$$v_3 = \begin{bmatrix} -c_2 c_3 s_4 + c_4 s_2 \\ -c_2 c_4 - c_3 s_2 s_4 \\ -s_3 s_4 \\ 0 \end{bmatrix}$$

$$v_4 = \begin{bmatrix} l_1 + l_2 s_2 + l_4 (-c_2 c_3 s_4 + c_4 s_2) \\ -c_2 c_4 l_4 - c_2 l_2 - c_3 l_4 s_2 s_4 \\ d_1 - l_4 s_3 s_4 \\ 1 \end{bmatrix}$$

IKBT can find all 8 positions of PUMA 560, and the solution graph shows the dependency among variables in Fig. 5. And these joint poses can be grouped into sets:

$$\begin{aligned}\text{Pose 1} &: [\theta_{1s1}, \theta_{2s4}, \theta_{3s1}, \theta_{4s8}, \theta_{5s7}, \theta_{6s7}] \\ \text{Pose 2} &: [\theta_{1s2}, \theta_{2s2}, \theta_{3s4}, \theta_{4s4}, \theta_{5s5}, \theta_{6s8}] \\ \text{Pose 3} &: [\theta_{1s1}, \theta_{2s1}, \theta_{3s2}, \theta_{4s2}, \theta_{5s3}, \theta_{6s6}] \\ \text{Pose 4} &: [\theta_{1s2}, \theta_{2s3}, \theta_{3s3}, \theta_{4s6}, \theta_{5s4}, \theta_{6s1}] \\ \text{Pose 5} &: [\theta_{1s2}, \theta_{2s3}, \theta_{3s3}, \theta_{4s5}, \theta_{5s8}, \theta_{6s2}] \\ \text{Pose 6} &: [\theta_{1s1}, \theta_{2s1}, \theta_{3s2}, \theta_{4s1}, \theta_{5s6}, \theta_{6s5}] \\ \text{Pose 7} &: [\theta_{1s1}, \theta_{2s4}, \theta_{3s1}, \theta_{4s7}, \theta_{5s2}, \theta_{6s4}] \\ \text{Pose 8} &: [\theta_{1s2}, \theta_{2s2}, \theta_{3s4}, \theta_{4s3}, \theta_{5s1}, \theta_{6s3}]\end{aligned}$$

#### 6.2.1 Result Verification

To verify the solution, we followed the process stated in section 5.1. The starting pose used is:

$$\theta_1 = 30^\circ, \theta_2 = 50^\circ, \theta_3 = 40^\circ,$$

$$\theta_4 = 45^\circ, \theta_5 = 120^\circ, \theta_6 = 60^\circ$$

as well as the parameters:

$$a_2 = 5, a_3 = 1, d_3 = 2, d_4 = 4$$

We got numerical T matrix:

$$T_d = \begin{bmatrix} -0.15720 & 0.97938 & 0.12682 & -1.68074 \\ -0.59374 & -0.19635 & 0.78032 & 1.33902 \\ 0.78914 & 0.04737 & 0.61237 & -4.83022 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We then plug these numbers into symbolic solutions obtained from last section, and get poses listed in Table 3. Specifically, Pose 7 is considered the same as initial input pose, with differences  $< 10^{-4}$ .

With the numerical poses, we computed forward kinematics for each pose. T matrix computed from Pose 1 is selected as example, as it shows the largest variation compared to the original T matrix:

Inverse kinematics solutions:

1)  $d_1$ , chosen solver: algebra

$$d_1 = Pz - l_4 r_{33}$$

2)  $\theta_2$ , chosen solver: sine or cosine

$$\theta_{2s1} = \text{asin}\left(\frac{1}{l_2} (Px - l_1 - l_4 r_{13})\right)$$

$$\theta_{2s2} = -\text{asin}\left(\frac{1}{l_2} (Px - l_1 - l_4 r_{13})\right) + \pi$$

1. Thanks to Prof. Melanie Shoemaker Plett.

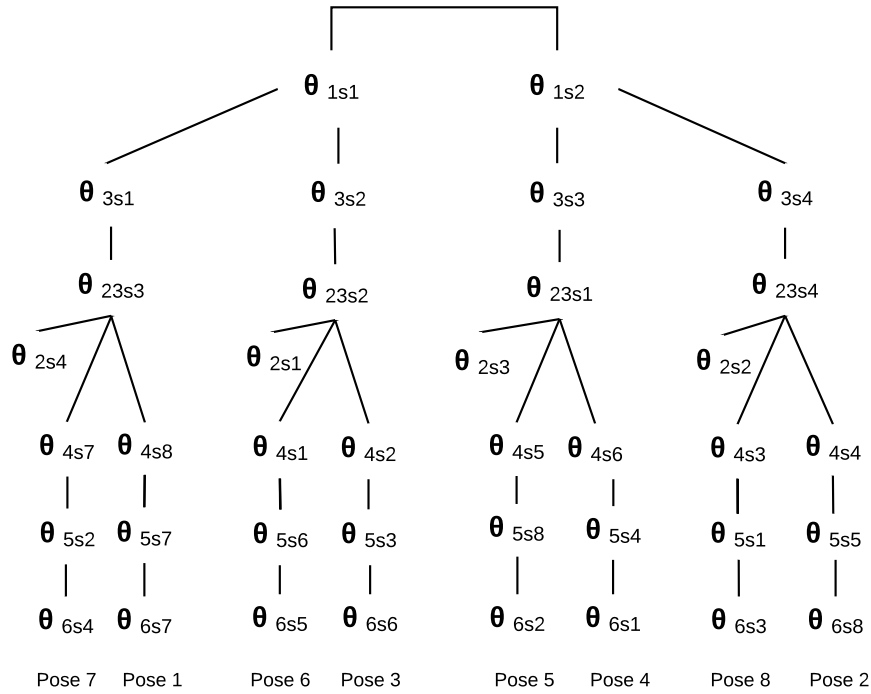


Fig. 5. PUMA 560 Solution Graph.

Solution Poses						
Pose	$\theta_1$	$\theta_2$	$\theta_3$	$\theta_4$	$\theta_5$	$\theta_6$
1	-287.08771	130.00008	-191.92745	-6.78054	-66.24462	4.13687
2	29.99995	49.99992	39.99994	-135.00007	-119.99981	-120.00010
3	29.99995	148.48625	-191.92745	142.01103	95.78709	-151.06756
4	-287.08771	31.51375	39.99994	18.00641	159.53838	18.33206
5	-287.08771	130.00008	-191.92745	173.21946	66.24462	-175.86313
6	29.99995	148.48625	-191.92745	-37.98897	-95.78709	28.93244
7	29.99995	49.99992	39.99994	44.99993	119.99981	59.99990
8	-287.08771	31.51375	39.99994	-161.99359	-159.53838	-161.66794

TABLE 3

Puma 560 Numerical Solutions

TABLE 4  
Chair Helper DH parameters

Link	$\alpha_{N-1}$	$a_{N-1}$	$d_N$	$\theta_N$
1	0	0	$d_1$	0
2	0	$l_1$	0	$\theta_2$
3	$\pi/2$	0	$l_2$	$\theta_3$
4	$\pi/2$	0	0	$\theta_4$
5	$-\pi/2$	0	$l_4$	$\theta_5$

3)  $\theta_3$ , chosen solver: tangent

$$\begin{aligned}\theta_{3s1} &= \text{atan2}(r_{33}, r_{13} \cos(\theta_{2s2}) + r_{23} \sin(\theta_{2s2})) \\ \theta_{3s2} &= \text{atan2}(-r_{33}, -r_{13} \cos(\theta_{2s2}) - r_{23} \sin(\theta_{2s2})) \\ \theta_{3s3} &= \text{atan2}(r_{33}, r_{13} \cos(\theta_{2s1}) + r_{23} \sin(\theta_{2s1})) \\ \theta_{3s4} &= \text{atan2}(-r_{33}, -r_{13} \cos(\theta_{2s1}) - r_{23} \sin(\theta_{2s1}))\end{aligned}$$

4)  $\theta_4$ , chosen solver: tangent

$$\begin{aligned}\theta_{4s1} &= \text{atan2}\left(-\frac{r_{33}}{\sin(\theta_{3s3})}, r_{13} \sin(\theta_{2s1}) - r_{23} \cos(\theta_{2s1})\right) \\ \theta_{4s2} &= \text{atan2}\left(-\frac{r_{33}}{\sin(\theta_{3s2})}, r_{13} \sin(\theta_{2s2}) - r_{23} \cos(\theta_{2s2})\right) \\ \theta_{4s3} &= \text{atan2}\left(-\frac{r_{33}}{\sin(\theta_{3s1})}, r_{13} \sin(\theta_{2s2}) - r_{23} \cos(\theta_{2s2})\right) \\ \theta_{4s4} &= \text{atan2}\left(-\frac{r_{33}}{\sin(\theta_{3s4})}, r_{13} \sin(\theta_{2s1}) - r_{23} \cos(\theta_{2s1})\right)\end{aligned}$$

5)  $\theta_5$ , chosen solver: tangent

$$\begin{aligned}\theta_{5s1} &= \text{atan2}\left(\frac{1}{\sin(\theta_{4s3})}(-r_{12} \sin(\theta_{2s2}) + r_{22} \cos(\theta_{2s2})), \frac{1}{\sin(\theta_{4s3})}(r_{11} \sin(\theta_{2s2}) - r_{21} \cos(\theta_{2s2}))\right) \\ \theta_{5s2} &= \text{atan2}\left(\frac{1}{\sin(\theta_{4s4})}(-r_{12} \sin(\theta_{2s1}) + r_{22} \cos(\theta_{2s1})), \frac{1}{\sin(\theta_{4s4})}(r_{11} \sin(\theta_{2s1}) - r_{21} \cos(\theta_{2s1}))\right) \\ \theta_{5s3} &= \text{atan2}\left(\frac{1}{\sin(\theta_{4s1})}(-r_{12} \sin(\theta_{2s1}) + r_{22} \cos(\theta_{2s1})), \frac{1}{\sin(\theta_{4s1})}(r_{11} \sin(\theta_{2s1}) - r_{21} \cos(\theta_{2s1}))\right) \\ \theta_{5s4} &= \text{atan2}\left(\frac{1}{\sin(\theta_{4s2})}(-r_{12} \sin(\theta_{2s2}) + r_{22} \cos(\theta_{2s2})), \frac{1}{\sin(\theta_{4s2})}(r_{11} \sin(\theta_{2s2}) - r_{21} \cos(\theta_{2s2}))\right)\end{aligned}$$

Solutions sets:

$$\begin{aligned}[d_1, \theta_{2s1}, \theta_{3s3}, \theta_{4s1}, \theta_{5s3}] \\ [d_1, \theta_{2s1}, \theta_{3s4}, \theta_{4s4}, \theta_{5s2}] \\ [d_1, \theta_{2s2}, \theta_{3s1}, \theta_{4s3}, \theta_{5s1}] \\ [d_1, \theta_{2s2}, \theta_{3s2}, \theta_{4s2}, \theta_{5s4}]\end{aligned}$$

Numerical verification confirmed that the inverse kinematics solutions are correct.

#### 6.4 Example Solution - Robot with strictly solution graph

The following example (Olson13) illustrates the necessity of a graph when tracking dependency, where variables have two independent parent variables, as shown in 6. DH parameters are listed in Table 5. Olson13 is a 6-DOF robot. The unknown variables are:  $[d_1 \ d_2 \ \theta_3 \ \theta_4 \ \theta_5 \ \theta_6]$ . The known parameters are:  $[l_3 \ l_4 \ l_5]$

TABLE 5  
Olson13 DH parameters

Link	$\alpha_{N-1}$	$a_{N-1}$	$d_N$	$\theta_N$
1	$-\pi/2$	0	$d_1$	$\pi/2$
2	$\pi/2$	0	$d_2$	$-\pi/2$
3	$\pi/2$	0	$l_3$	$\theta_3$
4	$\pi/2$	0	0	$\theta_4$
5	0	$l_4$	0	$\theta_5$
6	$\pi/2$	0	$l_5$	$\theta_6$

Forward kinematics:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & Px \\ r_{21} & r_{22} & r_{23} & Py \\ r_{31} & r_{32} & r_{33} & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix} = [v_1 \ v_2 \ v_3 \ v_4]$$

$$\begin{aligned}[v_1 \ v_2] &= \begin{bmatrix} -c_3s_6 + c_{45}c_6s_3 & -c_3c_6 - c_{45}s_3s_6 \\ -c_3c_{45}c_6 - s_3s_6 & c_3c_{45}s_6 - c_6s_3 \\ c_6s_{45} & -s_{45}s_6 \\ 0 & 0 \end{bmatrix} \\ [v_3 \ v_4] &= \begin{bmatrix} s_3s_{45} & c_4l_4s_3 + d_2 + l_5s_3s_{45} \\ -c_3s_{45} & -c_3c_4l_4 - c_3l_5s_{45} + d_1 \\ -c_{45} & -c_{45}l_5 + l_3 + l_4s_4 \\ 0 & 1 \end{bmatrix}\end{aligned}$$

The variables are solved in the following order:

1)  $\theta_3$ , chosen solver: tangent

$$\begin{aligned}\theta_{3s1} &= \text{atan2}(-r_{13}, r_{23}) \\ \theta_{3s2} &= \text{atan2}(r_{13}, -r_{23})\end{aligned}$$

2)  $\theta_4$ , chosen solver: sine or cosine

$$\begin{aligned}\theta_{4s1} &= \text{asin}\left(\frac{1}{l_4}(Pz - l_3 - l_5r_{33})\right) \\ \theta_{4s2} &= -\text{asin}\left(\frac{1}{l_4}(Pz - l_3 - l_5r_{33})\right) + \pi\end{aligned}$$

3)  $\theta_5$ , chosen solver: tangent

$$\begin{aligned}\theta_{5s1} &= \text{atan2}(r_{13} \sin(\theta_{3s2}) \cos(\theta_{4s1}) - r_{23} \cos(\theta_{3s2}) \cos(\theta_{4s1}) + r_{33} \sin(\theta_{4s1}), \\ &\quad r_{13} \sin(\theta_{3s2}) \sin(\theta_{4s1}) - r_{23} \sin(\theta_{4s1}) \cos(\theta_{3s2}) - r_{33} \cos(\theta_{4s1})) \\ \theta_{5s2} &= \text{atan2}(r_{13} \sin(\theta_{3s1}) \cos(\theta_{4s1}) - r_{23} \cos(\theta_{3s1}) \cos(\theta_{4s1}) + r_{33} \sin(\theta_{4s1}), \\ &\quad r_{13} \sin(\theta_{3s1}) \sin(\theta_{4s1}) - r_{23} \sin(\theta_{4s1}) \cos(\theta_{3s1}) - r_{33} \cos(\theta_{4s1})) \\ \theta_{5s3} &= \text{atan2}(r_{13} \sin(\theta_{3s2}) \cos(\theta_{4s2}) - r_{23} \cos(\theta_{3s2}) \cos(\theta_{4s2}) + r_{33} \sin(\theta_{4s2}), \\ &\quad r_{13} \sin(\theta_{3s2}) \sin(\theta_{4s2}) - r_{23} \sin(\theta_{4s2}) \cos(\theta_{3s2}) - r_{33} \cos(\theta_{4s2})) \\ \theta_{5s4} &= \text{atan2}(r_{13} \sin(\theta_{3s1}) \cos(\theta_{4s2}) - r_{23} \cos(\theta_{3s1}) \cos(\theta_{4s2}) + r_{33} \sin(\theta_{4s2}), \\ &\quad r_{13} \sin(\theta_{3s1}) \sin(\theta_{4s2}) - r_{23} \sin(\theta_{4s2}) \cos(\theta_{3s1}) - r_{33} \cos(\theta_{4s2}))\end{aligned}$$

4)  $\theta_6$ , chosen solver: tangent

$$\begin{aligned}\theta_{6s1} &= \text{atan2}(-r_{11} \cos(\theta_{3s2}) - r_{21} \sin(\theta_{3s2}), \\ &\quad -r_{12} \cos(\theta_{3s2}) - r_{22} \sin(\theta_{3s2})) \\ \theta_{6s2} &= \text{atan2}(-r_{11} \cos(\theta_{3s1}) - r_{21} \sin(\theta_{3s1}), \\ &\quad -r_{12} \cos(\theta_{3s1}) - r_{22} \sin(\theta_{3s1}))\end{aligned}$$

5)  $d_1$ , chosen solver: algebra

$$\begin{aligned}d_{1s1} &= Py + l_4 \cos(\theta_{3s2}) \cos(\theta_{4s1}) - l_5r_{23} \\ d_{1s2} &= Py + l_4 \cos(\theta_{3s1}) \cos(\theta_{4s1}) - l_5r_{23} \\ d_{1s3} &= Py + l_4 \cos(\theta_{3s2}) \cos(\theta_{4s2}) - l_5r_{23} \\ d_{1s4} &= Py + l_4 \cos(\theta_{3s1}) \cos(\theta_{4s2}) - l_5r_{23}\end{aligned}$$

6)  $d_2$ , chosen solver: algebra

$$\begin{aligned} d_{2s1} &= Px - l_4 \sin(\theta_{3s2}) \cos(\theta_{4s1}) - l_5 r_{13} \\ d_{2s2} &= Px - l_4 \sin(\theta_{3s1}) \cos(\theta_{4s1}) - l_5 r_{13} \\ d_{2s3} &= Px - l_4 \sin(\theta_{3s2}) \cos(\theta_{4s2}) - l_5 r_{13} \\ d_{2s4} &= Px - l_4 \sin(\theta_{3s1}) \cos(\theta_{4s2}) - l_5 r_{13} \end{aligned}$$

The following are the sets of joint solutions (poses) for this manipulator:

$$\begin{aligned} &[d_{1s3}, d_{2s3}, \theta_{3s2}, \theta_{4s2}, \theta_{5s3}, \theta_{6s1}] \\ &[d_{1s4}, d_{2s4}, \theta_{3s1}, \theta_{4s2}, \theta_{5s4}, \theta_{6s2}] \\ &[d_{1s1}, d_{2s1}, \theta_{3s2}, \theta_{4s1}, \theta_{5s1}, \theta_{6s1}] \\ &[d_{1s2}, d_{2s2}, \theta_{3s1}, \theta_{4s1}, \theta_{5s2}, \theta_{6s2}] \end{aligned}$$

The solution graph is shown in Fig. 6.  $d_1$ ,  $d_2$ , and  $\theta_5$  all share two independently solved parent variables:  $\theta_3$  and  $\theta_4$ . Thus, it results in a dependency graph.

## 7 DISCUSSION

### 7.1 Related Work and Comparison

Previous research on inverse kinematics lays a good foundation for IKBT. [3] used a rule-based pattern matching approach (implemented as an expert system in LISP) from which IKBT adapted the sin or cos solver, tangent solver, and simultaneous equation solver. Similar to IKBT, that system scanned a list of equations, and found the ones matching patterns, then fetching the respective solutions. Their method solved several commercial robots, including the more complicated PUMA 560. Limitations of this work include a hard coded framework for solution sequencing, and dependence on obsolescent software. Comparatively, IKBT uses very few rule-based solvers, indicative of more efficient logical reasoning.

[15] used a similar approach to ours. Their system also uses rule-based solvers implemented in LISP, though the detailed rules or the source code were not made public. The system of [15] solves equations sequentially and stops working on a variable as soon as it is solved. In contrast, IKBT's assigner node picks a variable first, and tries the entire toolbox for the chosen variable. We designed IKBT this way to get the optimal solution, in the situation where more than one solver applies to the same variable (choosing different equations). IKBT ranks the solutions obtained and chooses the best solution. [3], [15] did not show capability of finding all possible solution or tracking dependency among variables.

[4] used a different approach of converting the set of kinematics equations into a univariate polynomial using elimination techniques. It is effective in solving specific robots. However, whether their methods could be applied to other robots was not systematically tested. Also, because it uses an approach unlike what human experts do, it is harder to check the correctness of the solution or strategy: IKBT's toolbox contains only well known rules frequently used by human experts.

[16]'s solver used a product-of-exponentials formula, which doesn't require D-H parameters, and is robust in

dealing with kinematics singularities. However, it only showed the capability of handling numerical inputs and rendering numerical solutions, and very limited solving capability for complex robots (6-DOF robots  $\approx$  50%). Compared to [16], IKBT handles symbolic input, generates closed-form solutions, and achieved much better success rate with complex 5-DOF (100 %) and 6-DOF (80%) robots. [17] used evolutionary algorithms to get an approximate closed-form IK solution. By contrast, IKBT computes exact symbolic closed-form solutions.

IKFast performs symbolic inverse kinematics analysis as part of the OpenRAVE package [18]. Instead of general solving techniques, IKFast adapts a case-specific approach. It categories robots by their number of DOFs, and uses DOF-specific hard-coded algorithms for arms with different DOFs. IKFast generates a "dependency tree". However, a tree cannot represent the multiple independent dependencies we found for some variables in some robots. The graph-based representation generated by IKBT solves this problem. We have tried to run IKFast to compare the performance differences with IKBT, but we encountered several problems: First, it requires the full installation of the OpenRave suite. Second, its exclusive dependency on older version of software packages is incompatible with current versions. In order to run IKFast, one needs to downgrade Python and related packages.

One distinguishing feature of IKBT is the generation of solution graph of joint variables, though in some above-mentioned studies "dependency graph" was explored in other unrelated context. As described in details in section Solution Graph, dependency graph is essential to keep track of the joint poses, and make sure the solutions sets are complete and necessary. Complete means IKBT can find all possible joint poses if the solution exists. Necessary means that all solutions are unique, not duplicate to each other. Dependency tracking was usually done by engineers, to the best of our knowledge, IKBT is the first to automate this process.

### 7.2 Perspectives

Building on top of the previous research, IKBT has several defining advantages including applicability to any robot (up to 6-DOF), generalized solving scheme, extensible toolbox, modern and easy to implement language (Python), and dependencies limited to only a few libraries. We expect these characteristics will spur the wide adoption of IKBT into the robotics research and education communities.

Rule-based solvers included in IKBT's toolbox are commonly employed by human experts when solving inverse kinematics problems. This is advantageous because IKBT is not limited by robot configuration, specifically, it doesn't require three orthogonal axes in order to solve a robot.

IKBT's Behavior Tree represents an interpretable strategy - vital for judging many AI applications. This makes it easier to examine the correctness of the solution and the strategy formulating process. Although IKBT's approach costs more computing time than DOF-specific algorithms (4 ms, according to [18]), symbolic derivation only has to be done once per robot arm design. The Behavior Tree is easily modified and the solver toolbox is readily extensible.

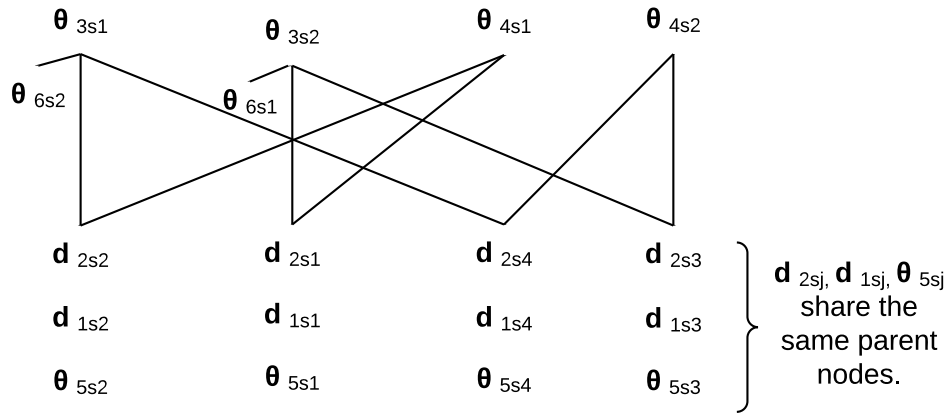


Fig. 6. Olson13 Solution Graph.

Although all the results presented here were generated by the BT of Fig. 2, it may be the case that a custom Behavior Tree could solve additional robots or solve robots more efficiently.

Behavior Tree has gained great success in game AI [19], [20], and showed substantial possibilities in robotics research [6], [8], [12], [21]. IKBT serves as a proof-of-concept of solving high-cognitive problems with Behavior Tree. IKBT mimics human experts' logical reasoning process, and constructs a generalized solving scheme applicable to an entire class of problems, using a small number of knowledge leaves. While most of current AI work focuses on recognizing and understanding scenarios, Behavior Tree emerges as a path to strengthen an equally vital component - logical reasoning. In IKBT, all knowledge-based solvers are coded by us, in other words, we "teach" the system about all the pre-existing rules and tricks people use when solving inverse kinematics problems. Given the current state of AI development, it is feasible to let the system to learn the knowledge by itself, through observing patterns and understanding the meaning behind. Combining forces of recognition, learning, and reasoning, we might be on our way to unlock the next level of autonomy.

## ACKNOWLEDGMENTS

We gratefully acknowledge support from National Science Foundation grant 1637444 and support for Blake Hannaford at Google-X / Google Life-Sciences / Verily in 2015.

## REFERENCES

- [1] P. I. Corke, "A robotics toolbox for matlab," *IEEE Robotics Automation Magazine*, vol. 3, no. 1, pp. 24–32, Mar 1996.
- [2] L. Kelmar and P. K. Khosla, "Automatic generation of forward and inverse kinematics for a reconfigurable modular manipulator system," *Journal of Robotic Systems*, vol. 7, no. 4, pp. 599–619, 1990. [Online]. Available: <http://dx.doi.org/10.1002/rob.4620070406>
- [3] L. G. Herrera-Bendezu, E. Mu, and J. T. Cain, "Symbolic computation of robot manipulator kinematics," in *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, Apr 1988, pp. 993–998 vol.2.
- [4] D. Halperin, "Automatic kinematic modelling of robot manipulators and symbolic generation of their inverse kinematics solutions (extended abstract)," pp. 310–317, 1991.
- [5] C.-U. Lim, R. Baumgarten, and S. Colton, "Evolving behaviour trees for the commercial game defcon," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2010, pp. 100–110.
- [6] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ogren, "Towards a unified behavior trees framework for robot control," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 5420–5427.
- [7] M. Colledanchise, M. Richard, and P. Ogren, "Synthesis of Correct-by-Construction Behavior Trees," in *Intelligent Robots and Systems (IROS 2017), 2017 IEEE/RSJ International Conference on*, Sept 2017, pp. 1482–1488.
- [8] M. Colledanchise, A. Marzinotto, D. V. Dimarogonas, and P. Ogren, "The advantages of using behavior trees in multi-robot systems," in *International Symposium on Robotics (ISR)*, June 2016.
- [9] M. Colledanchise, A. Marzinotto, and P. Ogren, "Performance analysis of stochastic behavior trees," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 3265–3272.
- [10] J. A. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, N. Pollard, M. Pivtoraiko, J. S. Valois, and R. Zhu, "An integrated system for autonomous robotics manipulation," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 2955–2962.
- [11] K. R. Guerin, C. Lea, C. Paxton, and G. D. Hager, "A framework for end-user instruction of a robot assistant for manufacturing," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 6167–6174.
- [12] D. Hu, Y. Gong, B. Hannaford, and E. J. Seibel, "Semi-autonomous simulated brain tumor ablation with ravenii surgical robot using behavior tree," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 3868–3875.
- [13] B. Hannaford, D. Hu, D. Zhang, and Y. Li, "Simulation results on selector adaptation in behavior trees," *CoRR*, vol. abs/1606.09219, 2016. [Online]. Available: <http://arxiv.org/abs/1606.09219>
- [14] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.
- [15] M. Wenz and H. Worn, "Solving the inverse kinematics problem symbolically by means of knowledge-based and linear algebra-based methods," in *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, Sept 2007, pp. 1346–1353.
- [16] I.-M. Chen and Y. Gao, "Closed-form inverse kinematics solver for reconfigurable robots," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 3, 2001, pp. 2395–2400 vol.3.
- [17] F. Chapelle and P. Bidaud, "A closed form for inverse kinematics approximation of general 6r manipulators using genetic programming," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 4, 2001, pp. 3364–3369 vol.4.

- [18] R. Diankov, "Automated construction of robotic manipulation programs," 2010.
- [19] M. Nicolau, D. Perez-Liebana, M. O'Neill, and A. Brabazon, "Evolutionary behavior tree approaches for navigating platform games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 3, pp. 227–238, Sept 2017.
- [20] A. Johansson and P. Dell'Acqua, "Emotional behavior trees," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2012, pp. 355–362.
- [21] P. Ogren, "Increasing modularity of uav control systems using computer game behavior trees," 08 2012.