Journal of Artificial Intelligence Research 65 (2019) 457-486 Submitted 01/2019; published 07/2019

IKBT: Solving Symbolic Inverse Kinematics with Behavior Tree

Dianmu Zhang DIANMUZ@UW.EDU
Blake Hannaford BLAKE@UW.EDU
Electrical and Computer Engineering
University of Washington
Seattle, WA, 98195 USA

Abstract

Inverse kinematics solves the problem of how to control robot arm joints to achieve desired end effector positions, which is critical to any robot arm design and implementations of control algorithms. It is a common misunderstanding that closed-form inverse kinematics analysis is solved. Popular software and algorithms, such as gradient descent or any multi-variant equations solving algorithm, claims solving inverse kinematics but only on the numerical level. While the numerical inverse kinematics solutions are relatively straightforward to obtain, these methods often fail, due to dependency on specific numerical values, even when the inverse kinematics solutions exist. Therefore, closed-form inverse kinematics analysis is superior, but there is no generalized automated algorithm. Up till now, the high-level logical reasoning involved in solving closed-form inverse kinematics made it hard to automate, so it's handled by human experts. We developed IKBT, a knowledge-based intelligent system that can mimic human experts' behaviors in solving closed-from inverse kinematics using Behavior Tree. Knowledge and rules used by engineers when solving closed-from inverse kinematics are encoded as actions in Behavior Tree. The order of applying these rules is governed by higher level composite nodes, which resembles the logical reasoning process of engineers. It is also the first time that the dependency of joint variables, an important issue in inverse kinematics analysis, is automatically tracked in graph form. Besides generating closed-form solutions, IKBT also explains its solving strategies in human (engineers) interpretable form. This is a proof-of-concept of using Behavior Trees to solve high-cognitive problems.

1. Introduction

Symbolic inverse kinematics analysis is a non-trivial task critical for operation and design of robot manipulators as well as animated characters. For example, in a simple serial robot (wrist robot) of three degrees of freedom (joint variables A, B, and C), the inverse kinematics computation takes the desired end-effector pose as input (typically as a homogeneous transform, left hand side of the equation), and solves for joint angles or joint displacements from the forward kinematic equations (right hand side of the equation).

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & Px \\ r_{21} & r_{22} & r_{23} & Py \\ r_{31} & r_{32} & r_{33} & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_1\cos\theta_2 & -\sin\theta_1\cos\theta_3 + \sin\theta_2\sin\theta_3\cos\theta_1 & \sin\theta_1\sin\theta_3 + \sin\theta_2\cos\theta_1\cos\theta_3 & 0 \\ \sin\theta_1\cos\theta_2 & \sin\theta_1\sin\theta_2\sin\theta_3 + \cos\theta_1\cos\theta_3 & \sin\theta_1\sin\theta_2\cos\theta_3 - \sin\theta_3\cos\theta_1 & 0 \\ -\sin\theta_2 & \sin\theta_3\cos\theta_2 & \cos\theta_2\cos\theta_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

The numerical solutions are often substituted for closed form symbolic solutions, where all elements in the desired end effector matrix (left hand side) are real numbers and solutions for joint variables are numerical values, using two major methods: A) gradient descent searches for a set of joint angles/length that minimize the cost function. Convergence of gradient descent can depend on the starting value and only generates one solution. Most existing software packages (Corke, 1996; Kelmar & Khosla, 1990) use a version of gradient descent as their core algorithm. The shared limitations include finding only one of the multiple solutions, convergence depending on the starting value, and problems with convergence near singular configurations. B) Solving multi-variant polynomial equations (Manocha & Canny, 1994; Murray, Sastry, & Zexiang, 1994). Though this method generates multiple possible solutions, it fails when the augmented transformation matrix is ill-conditioned, which is unavoidable in practice. And this method is often DOF-specific.

Comparatively, closed-form inverse kinematics analysis overcomes all these shortcomings of the numerical methods, but it's difficult to automate conceptually, because of the high-level mathematical reasoning needed. For example, in (1), above, closed-form inverse kinematics analysis for joint θ_1 begins by looking through all the equations and choosing two:

$$r_{21} = \sin\theta_1\cos\theta_2$$
$$r_{11} = \cos\theta_1\cos\theta_2$$

Though θ_2 is unsolved at this step, $cos(\theta_2)$ can be canceled out by dividing these two equations while calculating $atan2$. Depending on the value of $cos(\theta_2)$, θ_2 can have two solutions:

$$\begin{aligned} \theta_{1s1} &= \text{atan}_2(r_{21}, r_{11}), && \cos(\theta_2) > 0 \\ \theta_{1s2} &= \text{atan}_2(-r_{21}, -r_{11}), && \cos(\theta_2) < 0 \\ \theta_1 &\quad \text{undefined}, && \cos(\theta_2) == 0 \end{aligned}$$

Several groups have attempted to automate symbolic inverse kinematics analysis starting in the 1990's (Herrera-Bendezu, Mu, & Cain, 1988; Halperin, 1991), which laid the foundation for our work. Their work is reviewed and compared with ours in detail in the next section.

Though the sufficient conditions for the existence of closed-formed IK solutions is well-established (Pieper & Roth, 1969), the necessary condition remains unknown. For instance, a robot with three-intersecting axes has analytical IK solution; but it's not necessary for a robot to have three-intersecting axes to have closed-form IK solutions. In terms of degrees of freedom (DOF), any mechanism having greater than 6-DOF has unlimited number of analytical solutions, due to kinematic redundancy. IKBT solves cases up to 6-DOF.

In this work we develop an AI agent that solves closed-form inverse kinematics with the following goals: The agent

- Should solve closed-form inverse kinematics with a generalized algorithm applicable to most serial chain robot arms, without assumptions of configuration or degree-of-freedom
- Should explicitly use common knowledge that engineers use when solving the inverse kinematics problems, such as trig identities or the method of determinants, rather than relying on tricks for specific kinematic configurations
- Should have search and apply suitable knowledge or rules to equations containing unsolved joint variables as human experts do.
- Should be able to explain its solving strategy in an easy to interpret format
- Should be extensible and modifiable

To address these goals, we adapt Behavior Trees to construct an expert system, "IKBT", having the logical reasoning power to solve inverse kinematics symbolically without human supervision. A Behavior Tree - initially popular in video game AI, models intelligent agent behavior by incorporating specific tasks into action leaves (Lim, Baumgarten, & Colton, 2010; Marzinotto, Colledanchise, Smith, & Ogren, 2014; Colledanchise, Murray, & Ogren, 2017; Colledanchise, Marzinotto, Dimarogonas, & Ogren, 2016). 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 solution 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 and applied to equations and intermediate results by the Behavior Tree (see Transformations and Solvers section). The structure is readily extensible.
- IKBT generates a dependency *graph* of joint variables in the solutions, which specifies 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 often takes a human expert hours to complete.
- IKBT generates a report of its results and solution method in LaTeX, and generates code in Python and C++, creating functions which implement the derived solutions including domain (reachability) checking of numerical inputs (see Pose Validation section).
- Inverse kinematics solutions from IKBT are verifiable with numerical computations (facilitated by the IKBT code generator) (see Result Verification subsection under Results).
- Implementation in a modern open-source, cross-platform, programming language (Python). IKBT requires few dependencies outside of the standard Python distribution (mainly the symbolic manipulation package `sympy` and the unit testing framework `unittest`).

1.1 Related Work and Comparison

It is a common misconception that automated closed-form inverse kinematics is a solved problem. A few research groups introduced a generalized method to perform inverse kinematics analysis by solving multipolynomial multivariate equations (Manocha & Canny, 1994; Murray et al., 1994). However, the original research (Manocha & Canny, 1994), as well as authors of various textbooks, pointed out that this method involves a combination of numerical and symbolic manipulation (symbolic reduction at early steps, only auxiliary), and is thus not a closed-form solution. The multipolynomial method depends on numerical values in critical steps, such as eigenvalue computation. The resulting values for joint variables are only numerical. Other drawbacks of this method, as pointed out by Manocha and Canny (1994), include frequent poor conditioning of the matrices requiring inversion. Another explicit limitation of this method is that all joint variables need to be rotary - robots with one or more prismatic joints can't be solved by this method.

Previous research on closed-form inverse kinematics lays a good foundation for IKBT. One such example is a rule-based pattern matching approach (implemented as an expert system in LISP) (Herrera-Bendezu et al., 1988) 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, found the ones matching patterns, and then fetched 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. In comparison, IKBT uses only 6 rule-based solvers, indicative of more efficient combinatorial logical reasoning.

A similar approach implemented rule-based solvers in LISP (Wenz & Worn, 2007), although neither the detailed rules or the source code were made public. In this implementation, the system solved equations sequentially and stopped 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 from multiple equations). IKBT ranks the solutions obtained and chooses the best solution according

to specified criteria (described below). Unlike IKBT, previous research (Herrera-Bendezu et al., 1988; Wenz & Worn, 2007) did not show the ability to find all possible solutions or tracking dependencies among variables .

A different method used elimination techniques to convert the set of kinematic equations into a univariate polynomial (Halperin, 1991). 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. Each rule is provided with a straightforward unit test.

An additional solver used a product-of-exponentials formula, which doesn't require D-H parameters, and is robust in dealing with kinematics singularities (Chen & Gao, 2001). However, this solver 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 this D-H parameters-free solver (Chen & Gao, 2001), IKBT handles symbolic input, generates closed-form solutions, and achieved better success rate with complex 5-DOF (100 %) and 6-DOF (80%) robots. Another solver used evolutionary algorithms to get an approximate inverse solution (Chapelle & Bidaud, 2001). By contrast, IKBT computes exact symbolic closed-form solutions.

IKFast performs inverse kinematics analysis as part of the OpenRAVE package (Diankov, 2010). Instead of general solving techniques, IKFast adapts a case-specific hybrid approach.

We performed an in-depth source code analysis of IKFast which revealed the following major differences between IKFast and IKBT:

- Numerous lines of the IKFast source code specify the desired end effector position (the inverse kinematic input) as numerical values. Therefore, by definition, these results are not symbolic.

- Many numbers are presented in output equations and intermediate results of IKFast, where there should be only symbols in a closed form solution. The use of "iterations" in the IKFast solving code signals the underlying gradient descent/ascent method, which is prevalent in numerical IK process.

- Unlike IKBT, IKFast's results are complicated C++ code and not human understandable. Human interpretability is a major feature of IKBT.

- The output of IKFast is a `.cpp` file, which is not a symbolic solution. Even if C++ source code is claimed to be a symbolic solution, IKFast's code is too complex to manually verify. In addition to generating python and `.cpp` files, IKBT also generates LaTeXequations, obeying all standard conventions of a symbolic solution, that are compiled into human readable form with standard LaTeXtools.

Additional insights from IKFast code inspection reveal that IKFast categorizes robots by their number of DOFs, and uses hard-coded algorithms for arms with different DOFs. IKFast does generates a "dependency tree" like IKBT. However, a tree cannot represent

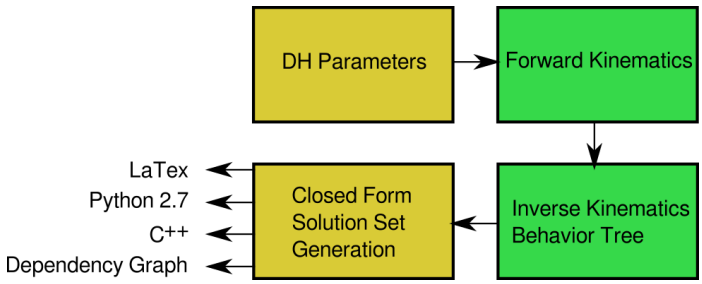

Figure 1: **Work Flow.** A Forward kinematics module computes symbolic kinematic equations to be solved ($T_d = T_s$) given the input DH parameters. Then the equations are evaluated for closed-form inverse kinematics solutions to each joint variable. Upon solving a robot, along with the solutions, a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

the multiple independent dependencies we found for some variables in some robots. The graph-based representation generated by IKBT solves this problem.

As described in detail in Section 4.4, a 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 duplicates of each other. Dependency tracking has previously been done manually. To the best of our knowledge, IKBT is the first to automate this process.

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 4x4 homogeneous transformation, T_0^6, is computed symbolically as

$$T_s = T_6^0 = T_1^0 T_2^1 T_3^2 T_4^3 T_5^4 T_6^5 \tag{2}$$

By convention, each transformation matrix takes a coordinate from the subscript frame and transforms it to the superscript frame (T_6^0 transforms from frame 6 to frame 0).

We denote the desired robot end effector pose as T_d (see left hand side of (1)). Then the inverse kinematics problem can be stated as solving

$$T_d = T_6^0(q_1, \ldots, q_6) \tag{3}$$

(where q_i are the unknown joint variables) for all sets of joint variables ($q_i = \theta_i$ or d_i) which satisfy 3. Related equations which can be used to find soluble equations include:

$$[T_1^0]^{-1} T_d = [T_1^0]^{-1} T_s \tag{4}$$

$$[T_2^1]^{-1}[T_1^0]^{-1}T_d = [T_2^1]^{-1}[T_1^0]^{-1}T_s \tag{5}$$

$$[T_{n-1}^{n-2}]^{-1}...[T_1^0]^{-1}T_d = [T_{n-1}^{n-2}]^{-1}...[T_1^0]^{-1}T_s \tag{6}$$

IKBT first symbolically calculates and simplifies these intermediate results (4 - 6) to augment (2). These intermediate results are stored in a buffer which is used as a lookup (joint variable - equations) table for solvers. 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. The individual equations are sorted 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 Behavior Tree's leaf nodes transform, or identify and solve, a particular kind of equation (see the Section 3). 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 θ_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 LaTeXreport, 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 (Marzinotto et al., 2014; Colledanchise, Marzinotto, & Ogren, 2014; Bagnell et al., 2012), collaborative robotics (Guerin, Lea, Paxton, & Hager, 2015; Colledanchise et al., 2016), and as a modeling language for intelligent robotic surgical procedures (Hu, Gong, Hannaford, & Seibel, 2015; Hannaford, Hu, Zhang, & Li, 2016).

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.

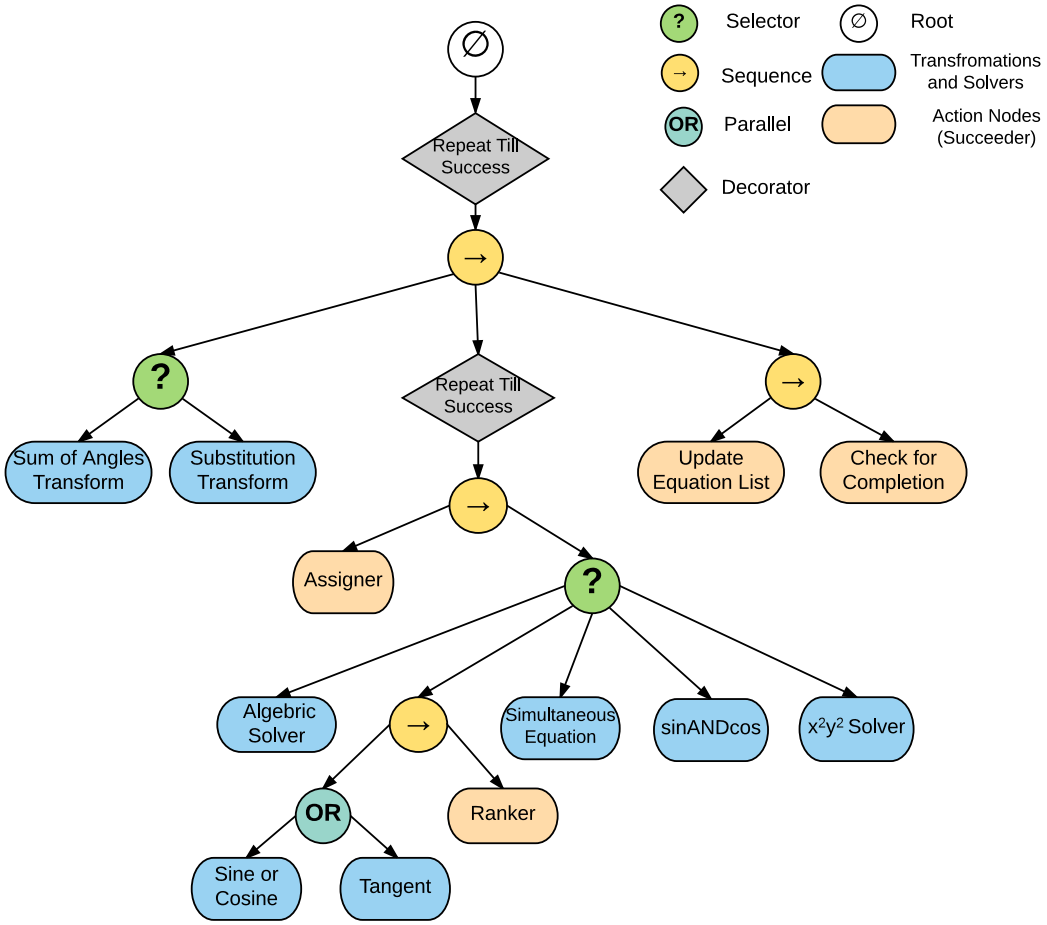

Figure 2: **IKBT Structure.** Node type explanation: Action nodes (leaves) carry out specific tasks, and returns SUCCESS or FAILURE. Succeeder is a special type of action nodes that only returns SUCCESS. Selector node ticks its children in turn, returns SUCCESS and stops if one of the children succeeds, 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.

3. Transformations and Solvers

In the following subsections, θ_i and d_i represent rotary 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

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

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

   Although the sum-of-angle simplification is done by sympy's `simplify` operation, creation of a new variable (θ_{xy}) is done by this node. This transformation also works for the sum of 3 angles (which allows solution of robot arms having 3 parallel sequential axes).

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

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

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

   The first equation can be transformed to:

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

   Eliminating one unknowns so that θ_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 returns 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. Solvers 1 - 4 are derived from straightforward algebra and trigonometry. 5-6 are adapted from respective literature (Herrera-Bendezu et al., 1988; Craig, 1989).

1. algebraic solver
   Identifies pattern

$$a + b\theta = c$$

   where $b \neq 0$. Solves for

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

as well as

$$a + bd_x = c$$

giving

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

2. sine or cosine solver
Identifies pattern

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

Solves for two solutions in each case:

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

and

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

3. tangent solver
identifies a pattern in two equations containing

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

If neither C_1 or C_2 contain unsolved variables:

$$\theta = \text{atan2}(aC_1, bC_2)$$

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:

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

$$\theta = \text{atan2}(aC, b) \quad C > 0$$

$$\theta = \text{atan2}(-aC, -b) \quad C < 0$$

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 two solutions:

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

   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 (Herrera-Bendezu et al., 1988) Identifies two equations:

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

   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 (Craig, 1989) :

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

   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

IKBT provides solution graph to track the dependency among joint variables. IKBT traces these dependencies in a mathematical sense, based on their symbolic solution equations. Intuitively, in the physical world, it can be interpreted as when one joint is set to a new value, the dependent joints values changes with it, in order to achieve certain end effector position and orientation.

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

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

in which θ_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 asin() operator, and additional multiplicity could come from the solution method for θ_{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.

To further illustrate the sources of multiplicity and dependency, we use the following example. The goal is to solve variables θ_1 and θ_2 from these equations:

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

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

From the first equation, θ_1 is solved as:

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

where we have used the subscript s to separate joint numbers from solution numbers. Here, θ_{1s2} means the second solution of θ_1. θ_1 has 2 solutions due to the nature of solver arccos(). Now that θ_1 is solved, we can solve θ_2 as:

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

θ_2 has 4 solutions, because its solver () generates 2 solutions, and on top of that, it θ_2 depends on the two solutions of θ_1. Example solution dependency is illustrated in 3.

In the resulting graph (shown in Fig. 3 a), each joint solution (e.g. θ_{2s1}, θ_{2s2}, etc.) is a node. A parent node is the node that appears in another node's solution expressions, in this example, θ_{1s1} is the parent of θ_{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 θ_5 has the following solutions:

$$\theta_{5s1} = \arccos(l + \cos(\theta_{4s1}))$$

$$\theta_{5s2} = -\arccos(l + \cos(\theta_{4s1}))$$

And θ_6 has solutions:

$$\theta_{6s1} = atan2(a + \cos(\theta_{4s1}), b + \sin(\theta_{5s1}))$$

$$\theta_{6s2} = atan2(a + \cos(\theta_{4s2}), b + \sin(\theta_{5s2}))$$

Though the solution of θ_6 involves both θ_4 and θ_5, its dependency to θ_4 is redundant. Given that θ_5 is also dependent on θ_4, the effects of choosing different θ_4 values (if applicable) on θ_6 are conveyed through θ_5. Therefore, when building a graph, only the edges between direct child-parents are added, in this case, Edge (θ_6, θ_5) and Edge(θ_5, θ_4). Shown in Fig. 3 b).

Classic 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 algorithms, 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, forming 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: [θ_{1s1}, θ_{2s1}], [θ_{1s1}, θ_{2s2}], [θ_{1s2}, θ_{2s3}], and [θ_{1s2}, θ_{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, θ_1 and θ_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 θ_3 depends on both θ_1 and θ_2:

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

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

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

5. Verification & Validation

This section answers two important questions: 1) How do we know if the IK solutions from IKBT are correct?, and 2) How to make sure that a requested end effector pose is valid, that IK solutions exist?

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 joint-space pose (note that joint limits are completed only after an IK solution is evaluated). Then we 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, we computed 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. The completeness of solutions is verified by comparing IKBT results to existing literature, and analyzed theoretically by tracing through joint dependency.

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:

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

or would require complex joint angles:

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

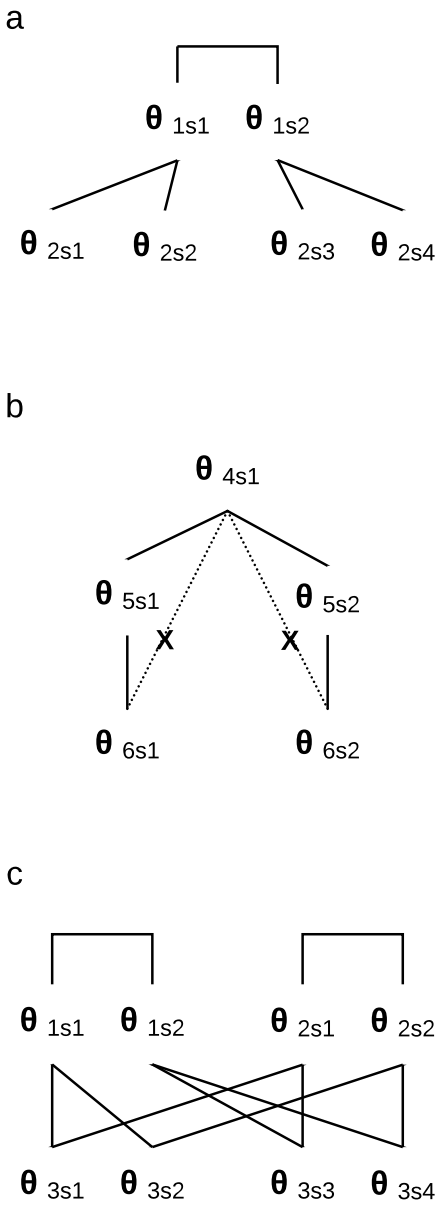

Figure 3: **Sample Solution Graph.** a) Simple example solution graph explains the origins of multiplicity. b) Redundancy pruning, 'x' marks the dependent relations that are not included in the graph. c) Example of a case of variables with multiple independent parents.

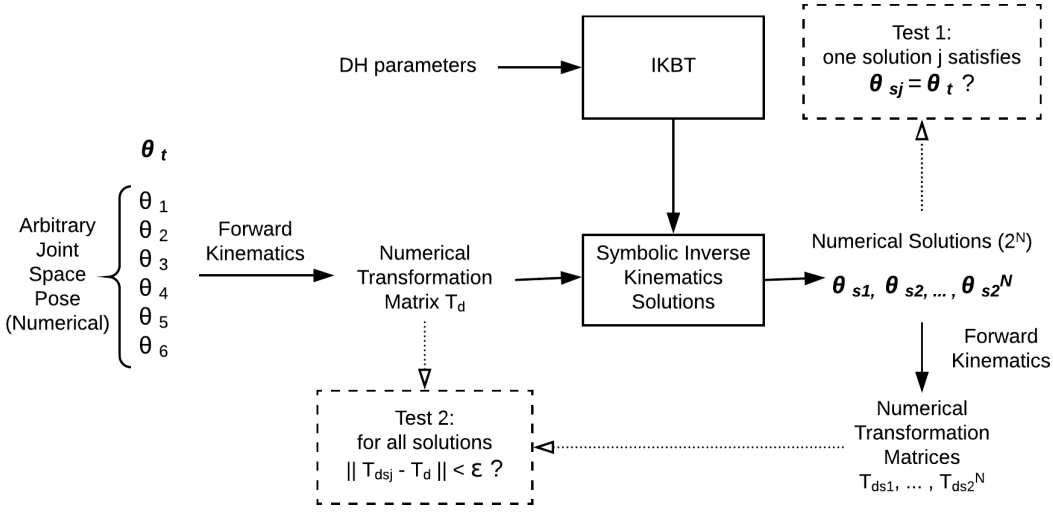

Figure 4: **Result Verification.** A numerical 4x4 homogeneous transformation matrix, T_d, is constructed from a reachable pose. Numerical joint space poses are computed from T_d using the closed-form solutions. For each solution pose, forward kinematics is calculated. The resulting transform matrices are compared against the original matrix, matching value is indicative of correct IKBT inverse kinematics analysis. 2^N indicates the number of solutions is always even.

Both the C++ and Python output modules of IKBT automatically 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. In practice, users can choose among all IK solutions that satisfies joint limits.

6. Experimental 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 Example 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. In all the solution results below, the equations are produced directly from the IKBT LaTeXoutput. A few edits have been made only to improve formatting at this column width.

Based on input of the Puma DH parameters, first, forward kinematics was calculated:

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

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

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

$$v_1 = \begin{bmatrix} c_6(-c_1 s_{23} s_5 + c_5(c_1 c_{23} c_4 + s_1 s_4)) - s_6(c_1 c_{23} s_4 - c_4 s_1) \\ c_6(c_5(-c_1 s_4 + c_{23} c_4 s_1) - s_1 s_{23} s_5) - s_6(c_1 c_4 + c_{23} s_1 s_4) \\ -c_6(c_{23} s_5 + c_4 c_5 s_{23}) + s_{23} s_4 s_6 \\ 0 \end{bmatrix}$$

$$v_2 = \begin{bmatrix} -c_6(c_1 c_{23} s_4 - c_4 s_1) - s_6(-c_1 s_{23} s_5 + c_5(c_1 c_{23} c_4 + s_1 s_4)) \\ -c_6(c_1 c_4 + c_{23} s_1 s_4) - s_6(c_5(-c_1 s_4 + c_{23} c_4 s_1) - s_1 s_{23} s_5) \\ c_6 s_{23} s_4 + s_6(c_{23} s_5 + c_4 c_5 s_{23}) \\ 0 \end{bmatrix}$$

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

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

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	α_{N-1}	a_{N-1}	d_N	θ_N
1	0	0	0	θ_1
2	$-\pi/2$	0	0	θ_2
3	0	a_2	d_3	θ_3
4	$-\pi/2$	a_3	d_4	θ_4
5	$\pi/2$	0	0	θ_5
6	$-\pi/2$	0	0	θ_6

where $c_1 = cos(\theta_1)$, $s_{23} = sin(\theta_2 + \theta_3)$ etc.
IKBT solved the six joint variables, $\theta_1 \dots \theta_6$ in the following order:

1. θ_1, chosen solver: sinANDcos

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

2. θ_3, chosen solver: x^2y^2

$$\begin{aligned}\theta_{3s1} &= \text{atan2}\,(-2a_2d_4, 2a_2a_3) + \text{atan2}(\sqrt{s - (t + (Px\cos(\theta_{1s1}) + Py\sin(\theta_{1s1}))^2)^2},\\ &\quad t + (Px\cos(\theta_{1s1}) + Py\sin(\theta_{1s1}))^2)\\ \theta_{3s2} &= \text{atan2}\,(-2a_2d_4, 2a_2a_3) + \text{atan2}(-\sqrt{s - (t + (Px\cos(\theta_{1s1}) + Py\sin(\theta_{1s1}))^2)^2},\\ &\quad t + (Px\cos(\theta_{1s1}) + Py\sin(\theta_{1s1}))^2)\\ \theta_{3s3} &= \text{atan2}\,(-2a_2d_4, 2a_2a_3) + \text{atan2}(\sqrt{s - (t + (Px\cos(\theta_{1s2}) + Py\sin(\theta_{1s2}))^2)^2},\\ &\quad t + (Px\cos(\theta_{1s2}) + Py\sin(\theta_{1s2}))^2)\\ \theta_{3s4} &= \text{atan2}\,(-2a_2d_4, 2a_2a_3) + \text{atan2}(-\sqrt{s - (t + (Px\cos(\theta_{1s2}) + Py\sin(\theta_{1s2}))^2)^2},\\ &\quad t + (Px\cos(\theta_{1s2}) + Py\sin(\theta_{1s2}))^2)\end{aligned}$$

where,

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

3. θ_{23}, chosen solver: simultaneous equation

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

4. θ_2, chosen solver: algebraic solver

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

5. θ_4, chosen solver: tangent

$$\begin{aligned}
\theta_{4s1} &= \text{atan2}(r_{13}\sin(\theta_{1s1}) - r_{23}\cos(\theta_{1s1}),\\
&\quad r_{13}\cos(\theta_{1s1})\cos(\theta_{23s2}) + r_{23}\sin(\theta_{1s1})\cos(\theta_{23s2}) - r_{33}\sin(\theta_{23s2}))\\
\theta_{4s2} &= \text{atan2}(-r_{13}\sin(\theta_{1s1}) + r_{23}\cos(\theta_{1s1}),\\
&\quad -r_{13}\cos(\theta_{1s1})\cos(\theta_{23s2}) - r_{23}\sin(\theta_{1s1})\cos(\theta_{23s2}) + r_{33}\sin(\theta_{23s2}))\\
\theta_{4s3} &= \text{atan2}(r_{13}\sin(\theta_{1s2}) - r_{23}\cos(\theta_{1s2}),\\
&\quad r_{13}\cos(\theta_{1s2})\cos(\theta_{23s4}) + r_{23}\sin(\theta_{1s2})\cos(\theta_{23s4}) - r_{33}\sin(\theta_{23s4}))\\
\theta_{4s4} &= \text{atan2}(-r_{13}\sin(\theta_{1s2}) + r_{23}\cos(\theta_{1s2}),\\
&\quad -r_{13}\cos(\theta_{1s2})\cos(\theta_{23s4}) - r_{23}\sin(\theta_{1s2})\cos(\theta_{23s4}) + r_{33}\sin(\theta_{23s4}))\\
\theta_{4s5} &= \text{atan2}(r_{13}\sin(\theta_{1s2}) - r_{23}\cos(\theta_{1s2}),\\
&\quad r_{13}\cos(\theta_{1s2})\cos(\theta_{23s1}) + r_{23}\sin(\theta_{1s2})\cos(\theta_{23s1}) - r_{33}\sin(\theta_{23s1}))\\
\theta_{4s6} &= \text{atan2}(-r_{13}\sin(\theta_{1s2}) + r_{23}\cos(\theta_{1s2}),\\
&\quad -r_{13}\cos(\theta_{1s2})\cos(\theta_{23s1}) - r_{23}\sin(\theta_{1s2})\cos(\theta_{23s1}) + r_{33}\sin(\theta_{23s1}))\\
\theta_{4s7} &= \text{atan2}(r_{13}\sin(\theta_{1s1}) - r_{23}\cos(\theta_{1s1}),\\
&\quad r_{13}\cos(\theta_{1s1})\cos(\theta_{23s3}) + r_{23}\sin(\theta_{1s1})\cos(\theta_{23s3}) - r_{33}\sin(\theta_{23s3}))\\
\theta_{4s8} &= \text{atan2}(-r_{13}\sin(\theta_{1s1}) + r_{23}\cos(\theta_{1s1}),\\
&\quad -r_{13}\cos(\theta_{1s1})\cos(\theta_{23s3}) - r_{23}\sin(\theta_{1s1})\cos(\theta_{23s3}) + r_{33}\sin(\theta_{23s3}))
\end{aligned}$$

6. θ_5, chosen solver: tangent

$$\theta_{5s1} = \text{atan2}(\frac{1}{\sin(\theta_{4s3})}(-r_{13}\sin(\theta_{1s2}) + r_{23}\cos(\theta_{1s2})),$$
$$-r_{13}\sin(\theta_{23s4})\cos(\theta_{1s2}) - r_{23}\sin(\theta_{1s2})\sin(\theta_{23s4}) - r_{33}\cos(\theta_{23s4})$$

$$\theta_{5s2} = \text{atan2}(\frac{1}{\sin(\theta_{4s7})}(-r_{13}\sin(\theta_{1s1}) + r_{23}\cos(\theta_{1s1})),$$
$$-r_{13}\sin(\theta_{23s3})\cos(\theta_{1s1}) - r_{23}\sin(\theta_{1s1})\sin(\theta_{23s3}) - r_{33}\cos(\theta_{23s3}))$$

$$\theta_{5s3} = \text{atan2}(\frac{1}{\sin(\theta_{4s2})}(-r_{13}\sin(\theta_{1s1}) + r_{23}\cos(\theta_{1s1})),$$
$$-r_{13}\sin(\theta_{23s2})\cos(\theta_{1s1}) - r_{23}\sin(\theta_{1s1})\sin(\theta_{23s2}) - r_{33}\cos(\theta_{23s2}))$$

$$\theta_{5s4} = \text{atan2}(\frac{1}{\sin(\theta_{4s6})}(-r_{13}\sin(\theta_{1s2}) + r_{23}\cos(\theta_{1s2})),$$
$$-r_{13}\sin(\theta_{23s1})\cos(\theta_{1s2}) - r_{23}\sin(\theta_{1s2})\sin(\theta_{23s1}) - r_{33}\cos(\theta_{23s1}))$$

$$\theta_{5s5} = \text{atan2}(\frac{1}{\sin(\theta_{4s4})}(-r_{13}\sin(\theta_{1s2}) + r_{23}\cos(\theta_{1s2})),$$
$$-r_{13}\sin(\theta_{23s4})\cos(\theta_{1s2}) - r_{23}\sin(\theta_{1s2})\sin(\theta_{23s4}) - r_{33}\cos(\theta_{23s4}))$$

$$\theta_{5s6} = \text{atan2}(\frac{1}{\sin(\theta_{4s1})}(-r_{13}\sin(\theta_{1s1}) + r_{23}\cos(\theta_{1s1})),$$
$$-r_{13}\sin(\theta_{23s2})\cos(\theta_{1s1}) - r_{23}\sin(\theta_{1s1})\sin(\theta_{23s2}) - r_{33}\cos(\theta_{23s2}))$$

$$\theta_{5s7} = \text{atan2}(\frac{1}{\sin(\theta_{4s8})}(-r_{13}\sin(\theta_{1s1}) + r_{23}\cos(\theta_{1s1})),$$
$$-r_{13}\sin(\theta_{23s3})\cos(\theta_{1s1}) - r_{23}\sin(\theta_{1s1})\sin(\theta_{23s3}) - r_{33}\cos(\theta_{23s3}))$$

$$\theta_{5s8} = \text{atan2}(\frac{1}{\sin(\theta_{4s5})}(-r_{13}\sin(\theta_{1s2}) + r_{23}\cos(\theta_{1s2})),$$
$$-r_{13}\sin(\theta_{23s1})\cos(\theta_{1s2}) - r_{23}\sin(\theta_{1s2})\sin(\theta_{23s1}) - r_{33}\cos(\theta_{23s1}))$$

7. θ_6, chosen solver: tangent

$$\theta_{6s1} = \text{atan2}(-\frac{-r_{12}\sin(\theta_{23s1})\cos(\theta_{1s2}) - r_{22}\sin(\theta_{1s2})\sin(\theta_{23s1}) - r_{32}\cos(\theta_{23s1})}{\sin(\theta_{5s4})}, \frac{-r_{11}\sin(\theta_{23s1})\cos(\theta_{1s2}) - r_{21}\sin(\theta_{1s2})\sin(\theta_{23s1}) - r_{31}\cos(\theta_{23s1})}{\sin(\theta_{5s4})})$$

$$\theta_{6s2} = \text{atan2}(-\frac{-r_{12}\sin(\theta_{23s1})\cos(\theta_{1s2}) - r_{22}\sin(\theta_{1s2})\sin(\theta_{23s1}) - r_{32}\cos(\theta_{23s1})}{\sin(\theta_{5s8})}, \frac{-r_{11}\sin(\theta_{23s1})\cos(\theta_{1s2}) - r_{21}\sin(\theta_{1s2})\sin(\theta_{23s1}) - r_{31}\cos(\theta_{23s1})}{\sin(\theta_{5s8})})$$

$$\theta_{6s3} = \text{atan2}(-\frac{-r_{12}\sin(\theta_{23s4})\cos(\theta_{1s2}) - r_{22}\sin(\theta_{1s2})\sin(\theta_{23s4}) - r_{32}\cos(\theta_{23s4})}{\sin(\theta_{5s1})}, \frac{-r_{11}\sin(\theta_{23s4})\cos(\theta_{1s2}) - r_{21}\sin(\theta_{1s2})\sin(\theta_{23s4}) - r_{31}\cos(\theta_{23s4})}{\sin(\theta_{5s1})})$$

$$\theta_{6s4} = \text{atan2}(-\frac{-r_{12}\sin(\theta_{23s3})\cos(\theta_{1s1}) - r_{22}\sin(\theta_{1s1})\sin(\theta_{23s3}) - r_{32}\cos(\theta_{23s3})}{\sin(\theta_{5s2})}, \frac{-r_{11}\sin(\theta_{23s3})\cos(\theta_{1s1}) - r_{21}\sin(\theta_{1s1})\sin(\theta_{23s3}) - r_{31}\cos(\theta_{23s3})}{\sin(\theta_{5s2})})$$

$$\theta_{6s5} = \text{atan2}(-\frac{-r_{12}\sin(\theta_{23s2})\cos(\theta_{1s1}) - r_{22}\sin(\theta_{1s1})\sin(\theta_{23s2}) - r_{32}\cos(\theta_{23s2})}{\sin(\theta_{5s6})}, \frac{-r_{11}\sin(\theta_{23s2})\cos(\theta_{1s1}) - r_{21}\sin(\theta_{1s1})\sin(\theta_{23s2}) - r_{31}\cos(\theta_{23s2})}{\sin(\theta_{5s6})})$$

$$\theta_{6s6} = \text{atan2}(-\frac{-r_{12}\sin(\theta_{23s2})\cos(\theta_{1s1}) - r_{22}\sin(\theta_{1s1})\sin(\theta_{23s2}) - r_{32}\cos(\theta_{23s2})}{\sin(\theta_{5s3})}, \frac{-r_{11}\sin(\theta_{23s2})\cos(\theta_{1s1}) - r_{21}\sin(\theta_{1s1})\sin(\theta_{23s2}) - r_{31}\cos(\theta_{23s2})}{\sin(\theta_{5s3})})$$

$$\theta_{6s7} = \text{atan2}(-\frac{-r_{12}\sin(\theta_{23s3})\cos(\theta_{1s1}) - r_{22}\sin(\theta_{1s1})\sin(\theta_{23s3}) - r_{32}\cos(\theta_{23s3})}{\sin(\theta_{5s7})}, \frac{-r_{11}\sin(\theta_{23s3})\cos(\theta_{1s1}) - r_{21}\sin(\theta_{1s1})\sin(\theta_{23s3}) - r_{31}\cos(\theta_{23s3})}{\sin(\theta_{5s7})})$$

$$\theta_{6s8} = \text{atan2}(-\frac{-r_{12}\sin(\theta_{23s4})\cos(\theta_{1s2}) - r_{22}\sin(\theta_{1s2})\sin(\theta_{23s4}) - r_{32}\cos(\theta_{23s4})}{\sin(\theta_{5s5})}, \frac{-r_{11}\sin(\theta_{23s4})\cos(\theta_{1s2}) - r_{21}\sin(\theta_{1s2})\sin(\theta_{23s4}) - r_{31}\cos(\theta_{23s4})}{\sin(\theta_{5s5})})$$

IKBT can find all 8 positions of PUMA 560, and the solution graph shows the dependency among variables in Fig. 5. These joint solutions are then grouped into sets corresponding

to correct poses:

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

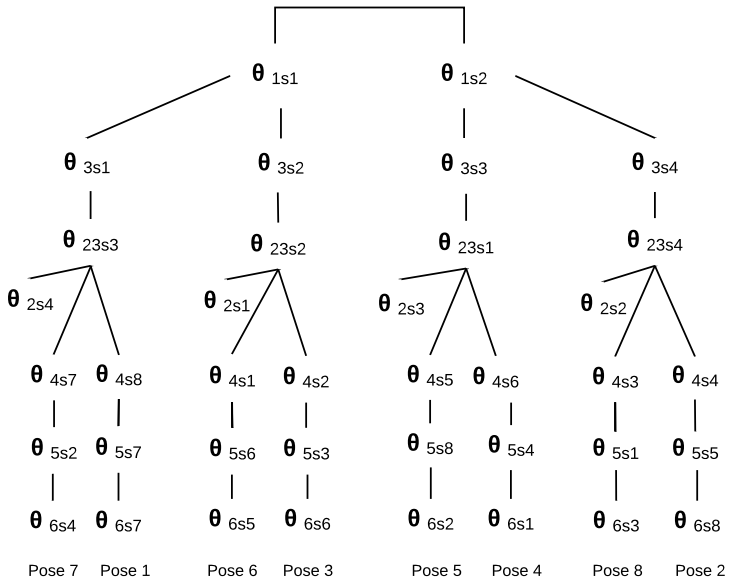

Figure 5: **PUMA 560 Solution Graph.**

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:

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

Solution Poses						
Pose	θ_1	θ_2	θ_3	θ_4	θ_5	θ_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**

From the DH parameters, the Forward kinematics code generated by IKBT generated the 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 plugged T_d into symbolic solutions obtained from the last section, and got the joint poses listed in Table 3. Note that Pose 7 (Table 3) is the same as initial input pose, within 10^{-4}.

With all the numerical poses, we computed forward kinematics. The T matrix computed from Pose 1 (T_{p1} is selected as an example, since it showed the largest variation compared to the original T matrix:

$$T_{p1} = \begin{bmatrix} -0.15720 & 0.97939 & 0.12682 & -1.68075 \\ -0.59374 & -0.19634 & 0.78033 & 1.33902 \\ 0.78915 & 0.04737 & 0.61237 & -4.83025 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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.

Note: these tests shown here is only a proof-of-principle. Extensive tests were carried out with numerous test values. Validity of end effector poses can be screened by pose validation method built-in IKBT, details see section Verification and Validation.

6.3 Example Solutions - 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)[1].

1. Thanks to Prof. Melanie Shoemaker Plett.

Table 4: Chair Helper DH parameters

Link	α_{N-1}	a_{N-1}	d_N	θ_N
1	0	0	d_1	0
2	0	l_1	0	θ_2
3	$\pi/2$	0	l_2	θ_3
4	$\pi/2$	0	0	θ_4
5	- $\pi/2$	0	l_4	θ_5

Forward kinematics:

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

$$v_1 = \begin{bmatrix} -c_2 s_3 s_5 + c_5(c_2 c_3 c_4 + s_2 s_4) \\ c_5(-c_2 s_4 + c_3 c_4 s_2) - s_2 s_3 s_5 \\ c_3 s_5 + c_4 c_5 s_3 \\ 0 \end{bmatrix} \qquad v_2 = \begin{bmatrix} -c_2 c_5 s_3 - s_5(c_2 c_3 c_4 + s_2 s_4) \\ -c_5 s_2 s_3 - s_5(-c_2 s_4 + c_3 c_4 s_2) \\ c_3 c_5 - c_4 s_3 s_5 \\ 0 \end{bmatrix}$$

$$v_3 = \begin{bmatrix} -c_2 c_3 s_4 + c_4 s_2 \\ -c_2 c_4 - c_3 s_2 s_4 \\ -s_3 s_4 \\ 0 \end{bmatrix} \qquad v_4 = \begin{bmatrix} l_1 + l_2 s_2 + l_4(-c_2 c_3 s_4 + c_4 s_2) \\ -c_2 c_4 l_4 - c_2 l_2 - c_3 l_4 s_2 s_4 \\ d_1 - l_4 s_3 s_4 \\ 1 \end{bmatrix}$$

IKBT gave the inverse kinematics solutions:

1. d_1, chosen solver: algebra

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

2. θ_2, chosen solver: sine or cosine

$$\theta_2 = \operatorname{atan2}\left(\frac{1}{l_2}(Px - l_1 - l_4 r_{13}), -\frac{1}{l_2}(Py - l_4 r_{23})\right)$$

3. θ_3, chosen solver: tangent

$$\theta_{3s1} = \operatorname{atan2}\left(r_{33}, r_{13}\cos(\theta_2) + r_{23}\sin(\theta_2)\right)$$
$$\theta_{3s2} = \operatorname{atan2}\left(-r_{33}, -r_{13}\cos(\theta_2) - r_{23}\sin(\theta_2)\right)$$

4. θ_4, chosen solver: tangent

$$\theta_{4s1} = \operatorname{atan2}\left(-\frac{r_{33}}{\sin(\theta_{3s2})}, r_{13}\sin(\theta_2) - r_{23}\cos(\theta_2)\right)$$
$$\theta_{4s2} = \operatorname{atan2}\left(-\frac{r_{33}}{\sin(\theta_{3s1})}, r_{13}\sin(\theta_2) - r_{23}\cos(\theta_2)\right)$$

5. θ_5, chosen solver: tangent

$$\theta_{5s1} = \text{atan2}\,(\frac{1}{\sin(\theta_{4s1})}(-r_{12}\sin(\theta_2) + r_{22}\cos(\theta_2)), \frac{1}{\sin(\theta_{4s1})}(r_{11}\sin(\theta_2) - r_{21}\cos(\theta_2)))$$
$$\theta_{5s2} = \text{atan2}\,(\frac{1}{\sin(\theta_{4s2})}(-r_{12}\sin(\theta_2) + r_{22}\cos(\theta_2)), \frac{1}{\sin(\theta_{4s2})}(r_{11}\sin(\theta_2) - r_{21}\cos(\theta_2)))$$

Solutions sets:

$$[d_1, \theta_2, \theta_{3s1}, \theta_{4s2}, \theta_{5s2}]$$
$$[d_1, \theta_2, \theta_{3s2}, \theta_{4s1}, \theta_{5s1}]$$

Numerical verification confirmed that the inverse kinematics solutions are correct.

6.4 Example Solutions - 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: $[d_1 \quad d_2 \quad \theta_3 \quad \theta_4 \quad \theta_5 \quad \theta_6]$. The known parameters are: $[l_3 \quad l_4 \quad l_5]$

Table 5: Olson13 DH parameters

Link	α_{N-1}	a_{N-1}	d_N	θ_N
1	$-\pi/2$	0	d_1	$\pi/2$
2	$\pi/2$	0	d_2	$-\pi/2$
3	$\pi/2$	0	l_3	θ_3
4	$\pi/2$	0	0	θ_4
5	0	l_4	0	θ_5
6	$\pi/2$	0	l_5	θ_6

Forward kinematics:

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

$$\begin{bmatrix} v_1 & v_2 \end{bmatrix} = \begin{bmatrix} -c_3 s_6 + c_{45} c_6 s_3 & -c_3 c_6 - c_{45} s_3 s_6 \\ -c_3 c_{45} c_6 - s_3 s_6 & c_3 c_{45} s_6 - c_6 s_3 \\ c_6 s_{45} & -s_{45} s_6 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} v_3 & v_4 \end{bmatrix} = \begin{bmatrix} s_3 s_{45} & c_4 l_4 s_3 + d_2 + l_5 s_3 s_{45} \\ -c_3 s_{45} & -c_3 c_4 l_4 - c_3 l_5 s_{45} + d_1 \\ -c_{45} & -c_{45} l_5 + l_3 + l_4 s_4 \\ 0 & 1 \end{bmatrix}$$

The variables are solved in the following order:

1. θ_3, chosen solver: tangent

$$\theta_{3s1} = \text{atan2}\left(-r_{13}, r_{23}\right)$$
$$\theta_{3s2} = \text{atan2}\left(r_{13}, -r_{23}\right)$$

2. θ_4, chosen solver: sine or cosine

$$\theta_{4s1} = \text{asin}\left(\frac{1}{l_4}\left(Pz - l_3 - l_5 r_{33}\right)\right)$$
$$\theta_{4s2} = -\text{asin}\left(\frac{1}{l_4}\left(Pz - l_3 - l_5 r_{33}\right)\right) + \pi$$

3. θ_5, chosen solver: tangent

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

4. θ_6, chosen solver: tangent

$$\theta_{6s1} = \text{atan2}(-r_{11}\cos\left(\theta_{3s2}\right) - r_{21}\sin\left(\theta_{3s2}\right), -r_{12}\cos\left(\theta_{3s2}\right) - r_{22}\sin\left(\theta_{3s2}\right))$$
$$\theta_{6s2} = \text{atan2}(-r_{11}\cos\left(\theta_{3s1}\right) - r_{21}\sin\left(\theta_{3s1}\right), -r_{12}\cos\left(\theta_{3s1}\right) - r_{22}\sin\left(\theta_{3s1}\right))$$

5. d_1, chosen solver: algebra

$$d_{1s1} = Py + l_4\cos\left(\theta_{3s2}\right)\cos\left(\theta_{4s1}\right) - l_5 r_{23}$$
$$d_{1s2} = Py + l_4\cos\left(\theta_{3s1}\right)\cos\left(\theta_{4s1}\right) - l_5 r_{23}$$
$$d_{1s3} = Py + l_4\cos\left(\theta_{3s2}\right)\cos\left(\theta_{4s2}\right) - l_5 r_{23}$$
$$d_{1s4} = Py + l_4\cos\left(\theta_{3s1}\right)\cos\left(\theta_{4s2}\right) - l_5 r_{23}$$

6. d_2, chosen solver: algebra

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

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

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

The solution graph is shown in Fig. 6. d_1, d_2, and θ_5 all share two independently solved parent variables: θ_3 and θ_4. Thus, it results in a dependency graph.

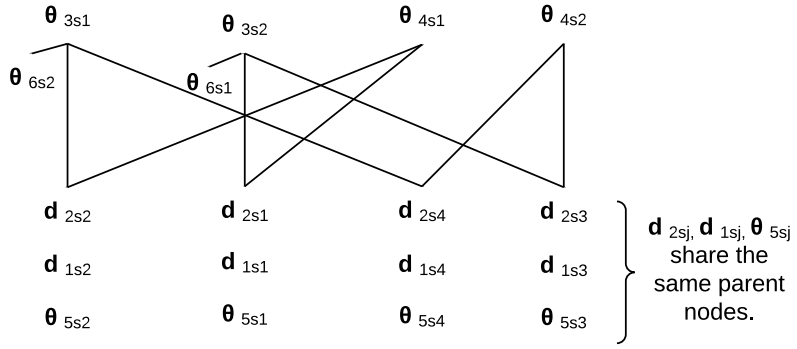

Figure 6: **Olson13 Solution Graph.**

7. Discussion

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

We reiterate the point of analytical inverse kinematics solutions is superior than numerical ones from introduction : numerical solvers often fails (unable to converge) due to dependency on specific numerical values, even when the inverse kinematics solutions exist - analytical solutions circumvent such dependency. To clarify, there are no numerical methods (with their attending difficulties) in our system, and purely symbolic reasoning is used. Historically, such reasoning has been proven difficult even when handled by human experts - to an extent that many commercial robots were designed with special configurations (such as three intersecting axes) to facilitate the solving process of closed-form inverse kinematics. IKBT, the automatic closed-form inverse kinematic solver, brings the possibility of broadening robot designs.

The rule-based solvers included in IKBT's toolbox are commonly employed by human experts when solving inverse kinematics problems. This is advantageous because IKBT solution methods are relateable to manual methods, and its approach is not limited by

robot configuration. Specifically, it doesn't require three orthogonal axes in order to solve a robot. One noteworthy characteristic about IKBT is that there is no upper limit for the number of solutions. As long as the solutions are valid and non-repetitive, IKBT can find all possible symbolic solutions for a robot. This is also due to the nature of IKBT - it is rule-based and not dependent on specific robot configuration.

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 (Diankov, 2010)), 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. 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.

The Behavior Tree representation has gained success in game AI (Nicolau, Perez-Liebana, ONeill, & Brabazon, 2017; Johansson & Dell'Acqua, 2012), and showed substantial possibilities in robotics research (Ogren, 2012; Marzinotto et al., 2014; Colledanchise et al., 2016; Hu et al., 2015). IKBT serves as a proof-of-concept of solving high-cognitive problems with Behavior Trees. 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 Trees emerge as a path to an equally vital component - combinatorial reasoning. Such logical reasoning enables AI agents to use limited knowledge base to solve problems of much larger magnitude. Intuitively, combinatorial reasoning is how human interact with the world: we decompose a unseen problem into solvable parts, then piece together modular knowledge to solve the larger problem. We don't get re-trained from scratch whenever we face new problems.

All knowledge-based solvers in IKBT are coded by us, in other words, we "teach" the system pre-existing rules and tricks people have used when solving inverse kinematics problems. Moving forward, the system can learn the knowledge by itself, by observing patterns and understanding their meaning. Learning in Behavior Trees has been explored in preliminary efforts(Lim et al., 2010; Dey & Child, 2013; Colledanchise, Nattanmai Parasuraman, & Ogren, 2018; Hannaford et al., 2016). Combining forces of learning (especially unsupervised) and combinatorial reasoning, we might be on our way to unlock the next level of autonomy.

Acknowledgments

We gratefully acknowledge support from National Science Foundation grant IIS-1637444 and support for Blake Hannaford at Google-X / Google Life-Sciences / Verily in 2015. We thank Moshe Lutz for proofreading this article.

Appendix A. Source Code and Reproducibility

Full source code and test case examples are available at Github: `https://github.com/uw-biorobotics/IKBT`.

References

Bagnell, J. A., et al. (2012). An integrated system for autonomous robotics manipulation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2955–2962.

Chapelle, F., & Bidaud, P. (2001). 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*, Vol. 4, pp. 3364–3369 vol.4.

Chen, I.-M., & Gao, Y. (2001). Closed-form inverse kinematics solver for reconfigurable robots. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, Vol. 3, pp. 2395–2400 vol.3.

Colledanchise, M., Murray, R., & Ogren, P. (2017). Synthesis of Correct-by-Construction Behavior Trees. In *Intelligent Robots and Systems (IROS 2017), 2017 IEEE/RSJ International Conference on*, pp. 1482–1488.

Colledanchise, M., Nattanmai Parasuraman, R., & Ogren, P. (2018). Learning of behavior trees for autonomous agents. *IEEE Transactions on Games*, 1–1.

Colledanchise, M., Marzinotto, A., Dimarogonas, D. V., & Ogren, P. (2016). The advantages of using behavior trees in multi-robot systems. In *International Symposium on Robotics (ISR)*.

Colledanchise, M., Marzinotto, A., & Ogren, P. (2014). Performance analysis of stochastic behavior trees. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3265–3272. IEEE.

Corke, P. I. (1996). A robotics toolbox for matlab. *IEEE Robotics Automation Magazine*, *3*(1), 24–32.

Craig, J. J. (1989). *Introduction to Robotics: Mechanics and Control* (2nd edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Dey, R., & Child, C. (2013). Ql-bt: Enhancing behaviour tree design and implementation with q-learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8.

Diankov, R. (2010). *Automated Construction of Robotic Manipulation Programs*. Ph.D. thesis, Carnegie Mellon University.

Guerin, K. R., Lea, C., Paxton, C., & Hager, G. D. (2015). A framework for end-user instruction of a robot assistant for manufacturing. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6167–6174. IEEE.

Halperin, D. (1991). Automatic kinematic modelling of robot manipulators and symbolic generation of their inverse kinematics solutions (extended abstract)..

Hannaford, B., Hu, D., Zhang, D., & Li, Y. (2016). Simulation results on selector adaptation in behavior trees. *CoRR*, *abs/1606.09219*.

Herrera-Bendezu, L. G., Mu, E., & Cain, J. T. (1988). Symbolic computation of robot manipulator kinematics. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pp. 993–998 vol.2.

Hu, D., Gong, Y., Hannaford, B., & Seibel, E. J. (2015). Semi-autonomous simulated brain tumor ablation with ravenii surgical robot using behavior tree. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3868–3875.

Johansson, A., & Dell'Acqua, P. (2012). Emotional behavior trees. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 355–362.

Kelmar, L., & Khosla, P. K. (1990). Automatic generation of forward and inverse kinematics for a reconfigurable modular manipulator system. *Journal of Robotic Systems*, *7*(4), 599–619.

Lim, C.-U., Baumgarten, R., & Colton, S. (2010). Evolving behaviour trees for the commercial game defcon. In *European Conference on the Applications of Evolutionary Computation*, pp. 100–110. Springer.

Manocha, D., & Canny, J. F. (1994). Efficient inverse kinematics for general 6r manipulators. *IEEE Transactions on Robotics and Automation*, *10*(5), 648–657.

Marzinotto, A., Colledanchise, M., Smith, C., & Ogren, P. (2014). Towards a unified behavior trees framework for robot control. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5420–5427. IEEE.

Murray, R. M., Sastry, S. S., & Zexiang, L. (1994). *A Mathematical Introduction to Robotic Manipulation* (1st edition). CRC Press, Inc., Boca Raton, FL, USA.

Nicolau, M., Perez-Liebana, D., ONeill, M., & Brabazon, A. (2017). Evolutionary behavior tree approaches for navigating platform games. *IEEE Transactions on Computational Intelligence and AI in Games*, *9*(3), 227–238.

Ogren, P. (2012). Increasing modularity of uav control systems using computer game behavior trees. In *2012 AIAA Guidance, Navigation, and Control Conference*.

Pieper, D., & Roth, B. (1969). The kinematics of manipulators under computer control. In *Proceedings of the Second International Congress on Theory of Machines and Mechanisms*, pp. 159–169.

Wenz, M., & Worn, H. (2007). 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)*, pp. 1346–1353.