EG UK Theory and Practice of Computer Graphics (2013)
Silvester Czanner and Wen Tang (Editors)

A Fast Inverse Kinematics Solver using Intersection of Circles

Srinivasan Ramachandran and Nigel W. John
School of Computer Science, Bangor University, UK

ABSTRACT

Kinematics is the study of motion without regard to the forces that causes it. In computer animation and robotics, Inverse Kinematics (IK) calculates the joint angles of an articulated object so that its end effector can be positioned as desired. This paper presents an efficient IK method using a geometric solver based on the intersection of circles. For an articulated object with n joints, our method will position the end-effector accurately and requires only a reverse iteration of (n-2). An intuitive user interface is provided, which automatically keeps the end effector between the maximum and minimum extent of the articulated object. Common problems that can occur with other IK methods are avoided. The algorithm has been implemented using WebGL and Javascript and tested by simulating a human hand, a three joint robot arm and human cycling motion, achieving interactive rates of up to 60 FPS

Categories and Subject Descriptors (according to ACM CCS): I.3. 7 [Computer Graphics]: Three-Dimensional Graphics and Realism, Animation.

1. Introduction

An articulated object is often modelled as a set of rigid segments connected with joints. Kinematics is an important area in robotics and computer animation when dealing with the motion of such objects and two approaches are commonly used. With Forward Kinematic (FK) the joint angles of the articulated object are specified and the subsequent position of the end effector(s) determined. This approach is simple but difficult to control the exact position of an end effector. With Inverse Kinematics (IK), the desired position of the end effector is specified and the joint angles are calculated so as to move the end effector as close as possible to this goal. IK solvers either attempt to determine an analytic solution (if it exists), or more commonly are iterative, and attempt to minimise the error between the current position and the desired position of the end effector [Par12]. Many IK approaches have been proposed but often they have a high computing cost or do not result in a satisfactory pose. In this paper we present a novel geometric method based on the intersection of circles. The aim is to demonstrate a solution which has low computing cost, is intuitive to use, and also produces poses that are visually realistic.

Section 2 presents the background and related work. Sections 3 and 4 then provide the detail of the algorithm, and results are presented in Section 5. The paper ends with conclusions and a discussion of future work.

2. Background and Related Work

Analytic solutions to the IK problem are not usually possible and so iterative solutions have been a topic of research for many years. Korein and Badler provided an early review of techniques [KB82], but there have been many other solutions proposed, for example [ZB94], [Buss04], [AL09]. Jacobian inverse methods, Heuristic methods, Pseudoinverse methods, quasi-Newton and conjugate gradient methods, and neural net and artificial intelligence methods have all been reported. A review of all IK approaches is outside the scope of this paper and we discuss just the most widely used techniques in current computer animation software.

2.1 Jacobian Solutions

A Jacobian matrix in vector calculus represents all first-order partial derivatives of a vector- or scalar-valued function with respect to another vector. It is used to help determine the best linear approximation to a differentiable function near a given point. For the IK problem, the Jacobian consists of a matrix of scalar values relating to the joint angles and the end-effector positions. The desired pose is iteratively calculated in a linear fashion. The inverse of the Jacobian must be determined and several techniques can be used to do this, providing an effective method of solving the IK problem, for example [WE84], [Wam86], [Bus04].

The computational cost involved means that Jacobian solutions are only suitable for real time applications that use a simple chain of joints. Further, if a situation arises such that the target for the end effector is unreachable, which is not uncommon, the system solution can start to oscillate about that target point. Then the previous stable pose must be used. There can also be singularity problems in the system leading to unnatural poses, which requires more flexible handling of constraints.

2.2 Heuristic Methods

Cyclic Coordinate Descent (CCD). CCD treats IK as a minimisation problem applied to each joint separately

DOI: 10.2312/LocalChapterEvents.TPCG.TPCG13.033-039

[WC91]. It is a fast technique that is well suited to computer games and other interactive applications. It has also been used to determine the configuration of protein chains [CD03].

The CCD algorithm serves to reduce errors related to position and rotation. Starting from the end-effector the iterator moves backwards to the base of the articulated object by manipulating one joint at a time, so that each time the end effector gets closer to the target. The number of times this operation is performed per iteration will always be equal to (n-1), where n is the number of joints.

Although fast, the CCD algorithm can often produce unrealistic animation with discontinuities in the resulting motion. CCD was developed for a serial chain type system and is not suitable for articulated objects with multiple end effectors.

Forward and Backward Reaching Inverse Kinematics (FABRIK). FABRIK is another fast iterative solver that gains efficiency by finding each joint position by locating a point on a line between the new and old position of the end effector, instead of using angle rotations. [AL11]. It uses previously calculated positions of the joints to find the updates to be applied in a forward and backward iterative mode. System error is minimised by adjusting each joint angle one by one. This process is carried out recursively until the end-effector is as close as possible to the target.

FABRIK has been demonstrated to produce more realistic poses than with CCD, without motion discontinuities. The algorithm can also manage multiple end-effectors. It is computationally efficient requiring a low number of iterations (although the exact number required cannot be determined in advance). FABRIK also determines the maximum reach of the articulated object and will calculate if the target is reachable before beginning the iterative cycle.

The heuristic approach has also been combined with a physics based method [HP12]. Here a mass-spring model allows force interaction between the masses to be used when solving the IK problem. They offer more parameters to fine tune motion (such as spring stiffness and damping) and still achieve adequate response times but not interactive.

3. Fundamentals

Our method is similar to the FABRIK approach and offers the same advantages. However, it is based on using the intersection of circles to calculate the orientation of rigid joints in the articulated object. The object is animated in 3D space but only 2D geometry calculations are required to compute the joint angles. It is also straightforward to calculate whether the desired target is reachable using our approach. If it is reachable, then when dealing with multiple joints there is a possibility that several different configurations can result in a solution. We will demonstrate that a visually acceptable solution will emerge from our method.

The basic geometry and data structures used by our method are summarised in this section.

3.1 Circle Intersection

When two circles intersect on a 2D plane (see Figure 1), then:

- There will exist only two intersection points.
- The intersection occurs along the circumference of the circles.
- The distance between the intersection points to their respective circle's origin will be a radius of the circle.

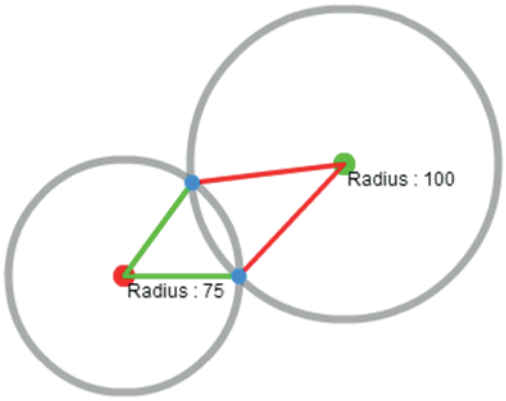

Figure 1: *Example of the intersection of two circles. The two intersection points lie along the circumference.*

3.2 Articulated Objects

The system is assumed to be composed of a chain of rigid links connected by joints. A tree data structure is used (Figure 2):

- The root joint is the base joint of the object and has no parent. Every joint except the root is connected by a link from exactly one other joint.
- Each joint is capable of having many child joints and there exists a unique path from the root to every other joint.
- End-effector(s) are the joint(s) that have to be moved to the target position. Joints along the path to the root need to be repositioned accordingly.

Either controlling the position of the joints or by controlling the rotation of the link the articulated object can be an animated. Table 1 summarises the data stored at each joint.

3.3 Axis of rotation

Figure 3 presents three orthographic projections onto a 2D plane for an example articulated object, where l_i are the links, j_i the joints, and $\Theta_{x/y/z}$ represent the axes of rotation when the links are rotated in 3D space. The rotation is performed about the perpendicular axis to the plane.

Table 1: Structure and properties of a Joint

Name	Type	Description
Radius	Float	The distance from its parent. Constant. Based on the two axes system (see below) this should be the width, height or the breadth values of the *joints* from its parent.
Total Radius	Float	The distance from the root. This distance must be maintained at all times.
Active radius	Float	The minimum distance that has to be maintained from the end-effector.
Current Radius	Float	The radius of the circle drawn from the root joint. This will change with every iteration.
Name	String	String representation of the path, which is also the name of the joint.
Is End Effector	Boolean	Flag for the end effector joint.
Parent	Joint	Refers to the parent joint. It is null if the joint is the root joint.
Children	List	List containing all the immediate children joints.
Location	Point	Stores the location of the joint as x and y float values.

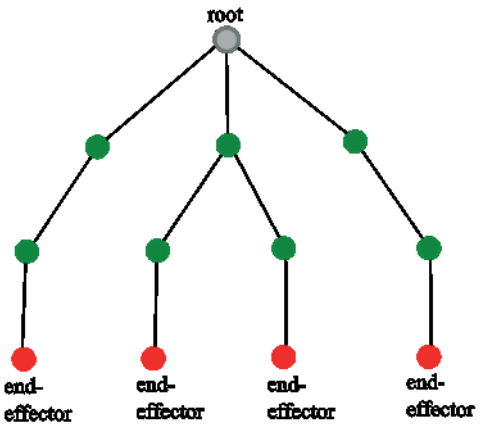

Figure 2: *Example of a tree structure for an articulated object with four possible end effectors.*

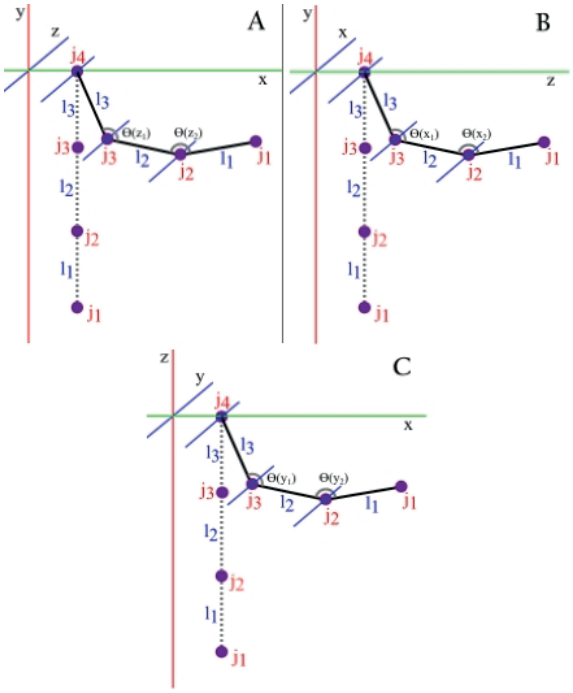

Figure 3: *Orthographic projections of an example articulated object. A: Rotation is about the z axis. B: Rotation is about the x axis. C: Rotation is about the y axis.*

4. Geometric IK Solver using Intersection of Circles

In this section, the proposed geometric IK solution is presented. Figure 4 summarises the main steps of how the algorithm operates for a simple example object composed of three links and a single end effector. Figure 4A shows the original configuration of the articulated object, set at its maximum extent. Each link l_1, l_2 and l_3 has a length r_1, r_2 and r_3, which are the distances between the respective nodes.

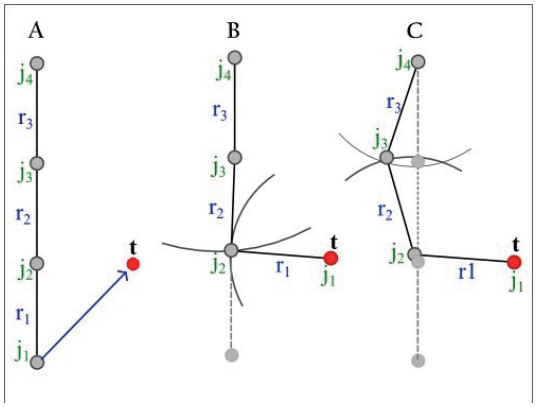

Figure 4: *A. Shows the initial configuration (set at the maximum extent) and the end-effector is to be positioned at the target location, t, if reachable. B. The first iteration. C. The second and final iteration. Hence for four joints two iterations are needed.*

Recall that the equation of a circle on a 2D plane is given by:

$$(\boldsymbol{x}-\boldsymbol{a})^2+(\boldsymbol{y}-\boldsymbol{b})^2=\boldsymbol{r}^2$$

where *(a,b)* is the centre of the circle, and *r* is the radius. We assume that the end effector can be placed at the target point, *t*, and then determine the two circles defined by:

1. The position of the base joint (j_4) with radius r_2+r_3
2. The target point with radius r_1

Solving for the intersection points of these two circles provides two solutions, and j_2 can be moved to either of them – refer to Figure 4B.

Figure 4C shows the configuration after the final iteration required to position all of the *joints* successfully (the example object has 4 joints; iterations required is total *joints* – 2 = 2). It requires only (n -2) iterations because the position of the base joint is stationary and the position of the end-effector is the location of the target **t** if reachable, and both can be excluded. Here, the two circles are defined by the new position of j_2 with radius r_2, and the position of j_4 with radius r_3. *Joint* j_3 is repositioned to one of the points of intersection.

In general, the *joints* are labelled $j_1, j_2, j_3, \ldots, j_n$, where j_n is the base joint and j_1 is the end-effector, The total length of the chain is defined by $L = \sum_1^{n-1} r_i$, where r_i is the radius (length) from j_i from its parent j_{i+1}. Also L is the maximum reachable distance from the *base* (a target location beyond this distance requires the *base* to be moved). Define *di* to be the distance to the target from the base:

$$di < L$$

If s_i is the arm length to the current joint (j_i):

$$s_i = \sum_{i=2}^{n-1} r_i$$

Define a_i to be the minimum distance to be maintained by j_i from j_1. The two circles of intersection required at each iteration are then:

$$(x-a)^2+(y-b)^2=(r_{i-1})^2 \qquad (1)$$

$$(x-c)^2+(y-d)^2=R^2 \qquad (2)$$

In circle (1) the location of the *child joint* is considered to be the origin, and its distance from the *joint* j_i , $r(_{i-1})$, as the radius. In circle (2), the *base joint* j_n is taken to be the origin and *R* is the radius. Note that:

$$r_i \le R \le \min(di - a_i\ ,\ s_i)$$

$$a = x(j_{i-1})$$

$$b = y(j_{i-1})$$

$$c = x(j_n)$$

$$d = y(j_n)$$

By solving the above equations in terms of '*x*' initially and substituting the result into (1) or (2), this will give the intersection points in terms of the Y axis. Repeat this process to obtain intersection values along the X axis. These can then be used to position the *joint* j_i. The process then continues backwards along the chain until the new position of *joint* j_{n-1} is found. The links are finally oriented by finding the angle between the located points.

Pseudocode of the key steps used by our IK solver is included in the appendix.

4.1 Safe Zones

A useful aspect of our method is the concept of Safe Zones:

- The **maximum safe zone** is the maximum reach of the articulated arm connected to the end effector, i.e. the total length to the end-effector when fully extended.
- The **minimum safe zone** is the position that can be reached by the end-effector so as to be as close as possible to the root joint. This relates to the possibility of the end effector being able to touch the base joint and is dependent on factors such as whether there are an odd or even number of links, and the difference in proportion of lengths between the links.

Other methods also take the maximum safe zone into account but few utilise a minimum safe zone. This is calculated in our method by initially locating the end-effector to the root joint location and applying the IK algorithm. This will fail if the distance between the root joint and the end-effector is not within an acceptable range, in which case the position of the end effector must be adjusted slightly. The process is iteratively performed until the end effector comes within an acceptable range. This is saved as the minimum acceptable distance to which the end-effector can reach towards the root joint.

Both safe zones are used as internal constraints when executing the IK algorithm. They prevent some of the discrepancies found with other IK approaches from occurring without the intervention of the animator.

5. Results

Three WebGL examples have been implemented to demonstrate the IK algorithm in action: a three joint robotic arm; a human cycling motion; and a more complex human hand grasping a ball. They are available online at:

http://www.vmg.cs.bangor.ac.uk/ik.php

These examples use 3D mesh models consisting of rigged components connected in tree structure.

Figure 5 shows a snapshot of a three link robotic arm being posed using our IK method. The 2D side view provides an intuitive user interface. The end effector (represented by the blue circle) can be positioned within the minimum and maximum safe zones by a mouse drag. The base of the robot is fixed to the floor. The resulting configuration of the robotic arm is instantaneously displayed in the 3D window. A top view is also provided to the animator (not shown) to change the orientation of the whole robot.

In Figure 6, a real time animation of a cycling motion has been implemented. There are two end effectors (the cyclist's feet) and they have been fixed to the pedals of the bicycle. The waist line contains the base joint. The pedals are being animated in a circular motion. The angles of the

knee and ankle joints are being calculated by our IK method. A natural cycling motion is obtained and rendered at 60 FPS on a standard laptop computer.

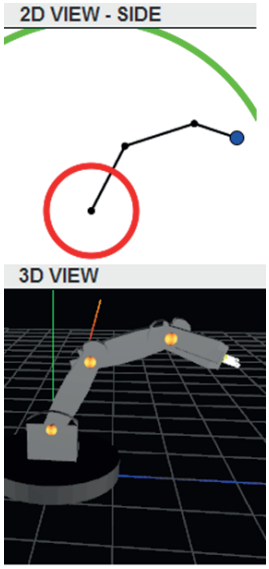

Figure 5: *Top image shows the 2D user interface panel that controls the position of the end effector. The bottom image shows the resulting pose of the 3D robotic arm.*

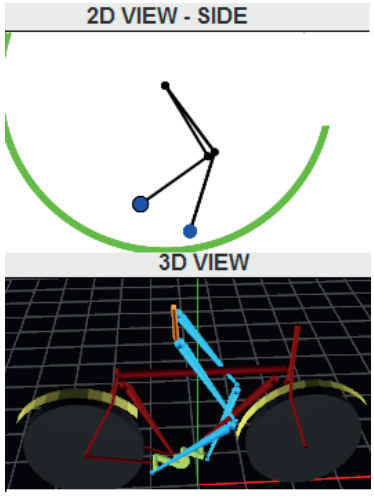

Figure 6: *Shows the 2D and 3D views of the human cyclist model. The pedaling motion is calculated and rendered in real time.*

Figure 7 shows the simulation of a more complex model, the human hand, using our algorithm. A ball is realistically grasped by the hand when it is in range. The IK problem is solved for each finger in real time.

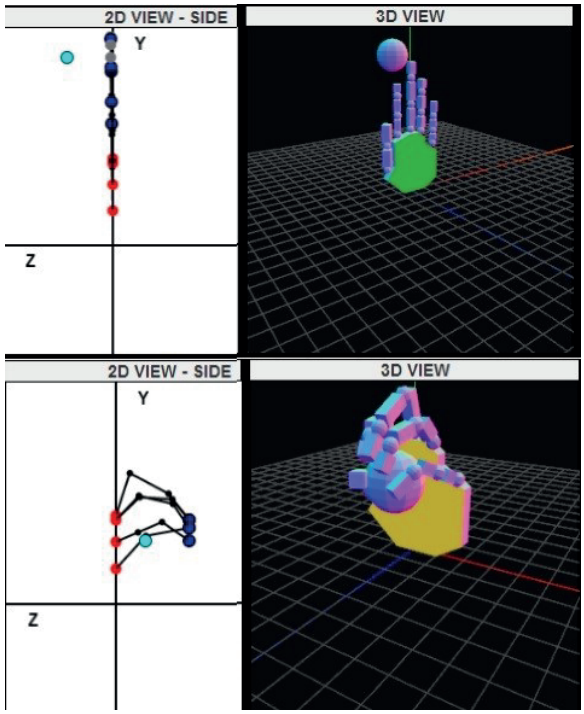

Figure 7 *The left images show the 2D user interface panel that controls the position of the ball (blue circle). The right images show the resulting pose of the 3D Human hand with multiple end effectors.*

These examples also demonstrate how an animator could use our IK technique for real time visualization of the posture of an articulated body, thus helping the animator to make better decisions about the animation sequences.

6. Conclusions and Future Work

We have described a new method of solving the IK problem utilizing the intersection of circles. The method takes advantage of the fact that the positions of the root joint and the end-effector are always known and uses a simple geometry technique to position interior joints. The results presented in the previous section indicate that we have at least matched other approaches such as the FABRIK algorithm in terms of being able to produce realistic motion without discontinuities. This is an advantage to gain from using a geometric approach as compared to a Jacobian solution. We have also demonstrated that our algorithm can easily manage multiple end-effectors. A detailed comparison with other widely used IK methods is currently being carried out.

A particular advantage of this algorithm is the use of Safe Zones to lessen the work of the animator. Other approaches succeed in reaching the target for the end effector but the desired pose is not always achieved. This is partly due to lack of consideration for the reachability of a target location, especially the minimum reach which is an important factor in a scenario where the end effector has to be at a point close to the base. The animator is often required to intervene and add extra constraints. Our approach mini-

mizes such intervention, however, and overlapping joints or links will never be produced.

The IK solver based on the intersection of circles is computationally efficient requiring a low number of iterations that is always equal to (n-2), where n is the number of joints in the system. It is therefore suitable for use in both the areas of computer animation and robotics where computing power may be limited. Additional implementations of the algorithm are currently in progress. We will make it available as a public domain plug-in for the Blender 3D computer graphics software project. We are also investigating its utility in robotics, and integrating it into the Robot Operating System (ROS).

References

[AL09] ARISTIDOU A., LASENBY, J.: Inverse Kinematics: a Review of Existing Techniques and Introduction of a New Fast Iterative Solver. *University of Cambridge Technical Report*, 2009

[AL11] ARISTIDOU A., LASENBY, J.: FABRIK: A Fast, Iterative Solver for the Inverse Kinematics Problem. *Graphical Models* 73 (5) (September 2011): 243–260.

[Bus04] BUSS S. R.: Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares Methods. *IEEE Journal of Robotics and Automation* 132, 4 (2004): 1–19.

[CD03] CANUTESCU A. A., DUNBRACK R. L.: Cyclic Coordinate Descent: A Robotics Algorithm for Protein Loop Closure. *Protein Science : a Publication of the Protein Society* 12 (5) (May 2003): 963–72. doi:10.1110/ps.0242703.

[HP12] HUNG J., PELACHAUD C.: An Efficient Energy Transfer Inverse Kinematics Solution. *In Proc. of MIG*, pp278-289, 2012

[KB82] KOREIN J. U., BADLER N. I.: Techniques for Generating the Goal-directed Motion of Articulated Structures. *IEEE Computer Graphics and Applications* 2, 9 (1982): 71–81.

[Par12] PARENT R.: *Computer Animation: Algorithms and Techniques, Third Edition*, Elsevier, 2012

[Wam86]WAMPLER, C.W.: Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *IEEE Transactions on Systems, Man and Cybernetics*, 16(1):93–101, 1986

[WC91] WANG L.-C. T., CHEN C.C.: A Combined Optimization Method for Solving the Inverse Kinematics Problems of Mechanical Manipulators. *IEEE Transactions on Robotics and Automation* 7, 4 (1991): 489–499. doi:10.1109/70.86079.

[WE84] WOLOVICH W., ELLIOTT H.: A Computatioal Technique for Inverse Kinematics." In *Proc. 23rd IEEE Conference on Decision and Control*, (1984) 1359–1363. doi:10.1109/CDC.1984.272258.

[ZB94] ZHAO J., BADLER N. I.: Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures. *ACM Transactions on Graphics (1994)*: 1–30

Acknowledgements

The algorithm was inspired by the FABRIK solution, which demonstrated that IK can be solved geometrically. We thank Dr Andreas Aristidou for having shared this work and his useful comments on this work.

Appendix 1: PseudoCode of the core IK routine

```
Int i;

For i is 1, i is less than path.size
minus 1, i increments by 1

  Initialise child to path.get  joint at
   i minus 1

  Call method
   currentNode.setCurrentRadius with
   ((currentNode.getTotalRadius plus
   currentNode.getActiveRadius) is
   greater than distance)

  If (child is not null) and
   (currentNode is not null)

        Initialise
        childToCurrentRadius to
        child.getRadius

        If (currentNode.getCurrentRadius
        is less than
        currentNode.getRadius)

              Set baseToCurrentRadius
              to currentNode.getRadius

        EndIf

        Set points to
        Math2.circleIntersections with
        childToCurrentRadius,
baseToCurrentRadius, child location and
base location

        If (points.get with 0 is
        not equal to null)
        and (points.get with 1
        is not equal to null)

              Set point to
              points.get with 0

        EndIf

EndIf

EndFor
```