Journal of
open research software

Lansley, A et al 2016 Caliko: An Inverse Kinematics Software Library Implementation of the FABRIK Algorithm. *Journal of Open Research Software,* 4: e36, DOI: http://dx.doi.org/10.5334/jors.116

SOFTWARE METAPAPER

Caliko: An Inverse Kinematics Software Library Implementation of the FABRIK Algorithm

Alastair Lansley, Peter Vamplew, Philip Smith and Cameron Foale

Federation University Australia, AU

Corresponding author: Alastair Lansley
(a.lansley@federation.edu.au)

The Caliko library is an implementation of the FABRIK (Forward And Backward Reaching Inverse Kinematics) algorithm written in Java. The inverse kinematics (IK) algorithm is implemented in both 2D and 3D, and incorporates a variety of joint constraints as well as the ability to connect multiple IK chains together in a hierarchy. The library allows for the simple creation and solving of multiple IK chains as well as visualisation of these solutions. It is licensed under the MIT software license and the source code is freely available for use and modification at: https://github.com/feduni/caliko

Keywords: Inverse; Kinematics; IK; FABRIK; Software; Library; Robotics; Leap; Java

(1) Overview

Introduction

The Caliko software library is a free open-source software (FOSS) implementation of the FABRIK inverse kinematics (IK) algorithm [1]. The library was created to provide IK solutions for hand and finger location data provided by a Leap Motion sensor [2].

Although the Leap Motion controller provides IK solutions for finger/hand poses, it does so via the sensor's proprietary closed-source driver, where there is no knowledge or discussion of the algorithm(s) used or ability to modify any IK parameters. The Caliko library can be used with the Leap Motion controller or indeed any collection of connected point data to perform the same IK solving functions, but in an open/transparent fashion with full access to the source code and numerous tuneable parameters.

The Caliko library is currently being used to implement novel HCI input methods, which are yet to be published, but the library itself is freely available. A demonstration video that outlines the setup and available functionality can be seen on YouTube [3].

Implementation and architecture

In order to understand the software structure, it makes sense to first understand the nature of IK problems – where an articulated body is modelled using a series of '***bones***' which have start and end locations in 2D or 3D space. Multiple bones may be connected together, so that the end location of one bone is the same as the start location of the next bone. These connected bones form an IK '***chain***'.

Each bone has a single '***joint***' which may be considered to be at the start location of the bone, and joints may constrain the allowable movement of bones with regard to the previous bone in the IK chain, or with regard to an arbitrary direction.

To easily work with multiple IK chains that share the same target location (and which may or may not be connected to each other), a final holder called a '***structure***' is provided. All IK chains in a structure can be solved by simply calling:

```
structure.solveForTarget(some_target_location);
```

The goal of *solving an IK chain* is then one that may be stated as: given an IK chain (which may or may not have a 'fixed' start location), what is the best configuration of the chain so that its end-effector (i.e. the tip of the final bone) can reach a given target location, or get as close as possible if the chain cannot successfully be solved for distance? Further details on the nature of inverse kinematics along with a variety of IK techniques and methods can be found in [5] and [6].

As a basic example, a simple 2D IK chain containing three bones may be constructed and solved using the Caliko library as shown in **Figure 1** below. Further details regarding the Caliko classes and their usage can be found in the user and technical documentation provided in the **doc** folder of any release of the Caliko software.

The Caliko library was written over a period of 18 months in the Java programming language and is single threaded, but the standard Java threading mechanisms can be used to easily allow each thread to process any 'owned' IK chains or structures concurrently. Pre-compiled releases of the library may be downloaded from its GitHub source code repository, or the source code itself can be downloaded and packaged into a release using the Maven

(a)

```
Vec2f RIGHT = new Vec2f(1.0f, 0.0f);
Vec2f targetLocation = new Vec2f(15.0f, 12.0f);
float boneLength = 10.0f;

// Create a chain with three bones
FabrikChain2D chain = new FabrikChain2D();
FabrikBone2D base = new FabrikBone2D(new Vec2f(), RIGHT, boneLength);
chain.addBone(base);
chain.addConsecutiveBone(RIGHT, boneLength);
chain.addConsecutiveBone(RIGHT, boneLength);

// Solve the chain for the given target location
chain.solveForTarget(targetLocation);
```

(b)

Figure 1: Creating and solving a basic 2D chain. **(a)** shows the initial configuration of the chain and **(b)** its solved state. The yellow square represents the chain's target location.

build management system [4] by issuing the following command from the top-level directory of the library:

```
mvn package
```

As well as implementing the 'core' FABRIK algorithm to solve IK chains, the Caliko library allows for the accuracy of solutions to be traded against computational effort via the following mechanisms:

- **Solve distance**
  - The minimum distance between the *end-effector* and the target. This can be altered to be within varying distance thresholds before accepting the solution. For example, if working in 2D with screen-space directly mapped to world-space, a solve distance of less than 0.5 (i.e. half a pixel) would provide a visually identical solution to one which was accurate down to 0.001 pixels, but without the potential need for additional iterative solve attempts.
- **Iteration change**
  - The minimum change in solve distance across consecutive solve attempts. If a chain cannot be solved and/or progress being made towards the solution is below a given threshold then Caliko can dynamically abort rather than expend computational effort on a solution which is only fractionally better than the current solve distance, and
- **Iteration count**
  - The maximum number of solve attempts to be made. As the FABRIK algorithm iteratively improves upon solutions, a limit may be placed upon the maximum number of iterations to attempt before deciding that enough computational effort has been expended.

The FABRIK algorithm is generally very successful in solving an IK chain, however there may be times when a chain cannot be successfully solved for distance due to a variety of factors, such as that the target might be further away than the length of the chain, or that the chain may be over-constrained. It is under these circumstances where little can be done beyond the current solution that dynamically aborting what is essentially a lost cause helps to minimise unnecessary processing, and where the current best solution may be accepted.

The software itself is broken up into 3 main packages as shown in **Figure 2** below:

- **au.edu.federation.caliko**
  - This contains the FabrikBone, FabrikJoint, FabrikChain and FabrikStructure classes in both 2D and 3D versions.

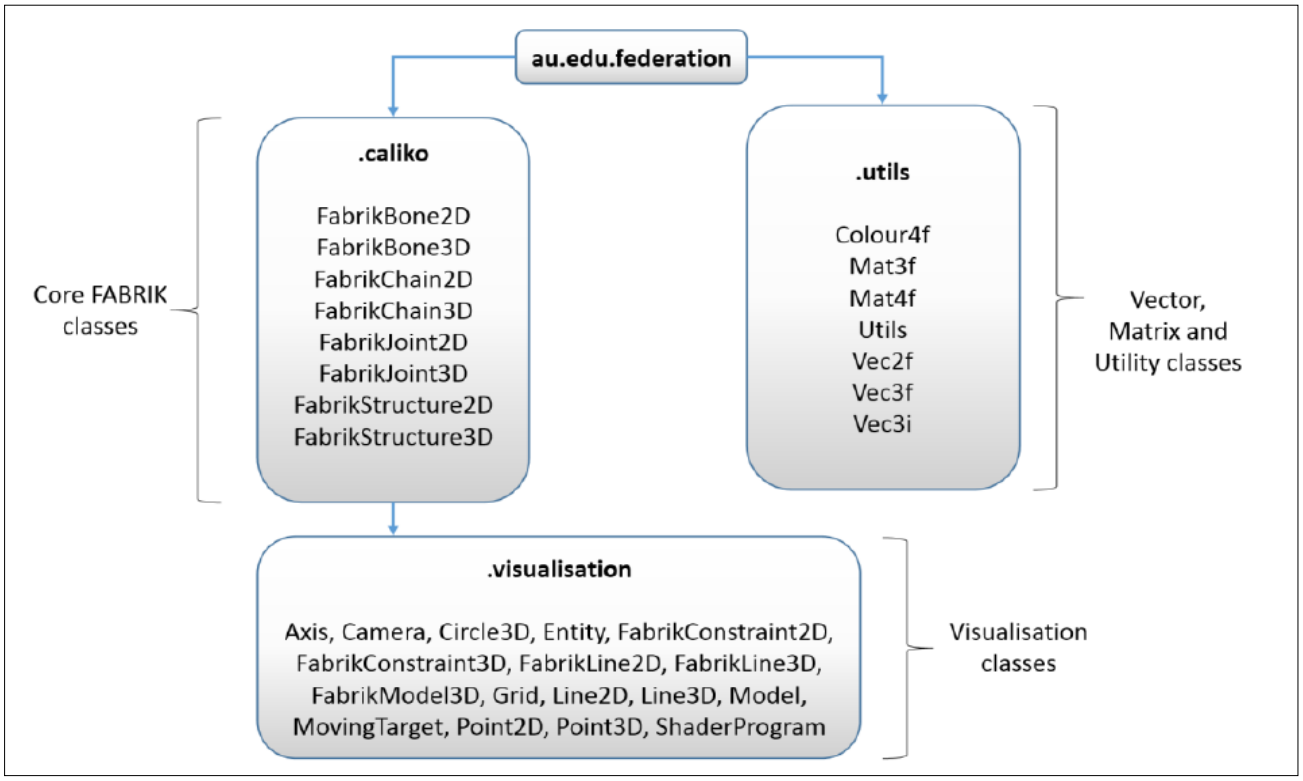

Figure 2: The Caliko library classes and package structure.

- **au.edu.federation.caliko.visualisation**
  - This contains classes which allow for easy visualisation of IK bones, joints, chains and structures using OpenGL 3.3 and GLSL version 330.
- **au.edu.federation.utils**
  - This contains custom 3x3 and 4x4 matrix classes, as well as some vector classes and a Utils class with a number of helper functions.

The demonstration application that comes with the Caliko library has an additional **au.edu.federation.alansley** package which contains a main function that sets up the 2D and 3D demonstration examples and creates an OpenGL window with relevant mouse and keyboard handlers to allow for user input.

Quality control

The software has been functionally tested extensively during creation on both the Windows and Linux platforms.

FABRIK is a computationally inexpensive IK algorithm when compared to alternatives such a Cyclic Coordinate Descent (CCD) or Jacobian methods – and in addition, it provides natural and continuous solutions to IK problems which do not exhibit discontinuities [1].

In execution, and due to the nature of the FABRIK algorithm and the user-controllable dynamic-abort settings implemented in the Caliko library, processor usage is typically very low. For example using the Oracle HotSpot JVM (Java Virtual Machine) the Java process may use around ~1–2% of a single core on an Intel i7 processor whilst actively solving and displaying multiple IK chains with approximately a dozen constrained bones in each. When calculating but not graphically displaying the solutions, processor usage is lower still. A graph of solve time against number of bones in an IK chain is shown below in **Figure 3**. As can be seen, the solve duration exhibits close to a linear response to the number of bones in the IK chain which makes it suitable for real-time usage scenarios. The source code used to generate these figures can be found in the **perf-test** folder of the Caliko GitHub repository.

The Caliko library is designed to run without visualising the results of its work, that is, it can simply run the algorithm and return the updated IK chain configuration to the user. Visualisation functionality is provided, but is intended only as a quick and easy method of visualising and experimenting with IK chains and structures.

When constraining bones in an IK chain, solutions can exhibit discontinuities when they get stuck against a constraint and then 'pull-through' from one side to the other. However, this is not necessarily a problem per se, it's simply the result of the new solution improving upon the old solution (that got stuck against a constraint) by solving the chain 'the other way around'.

Local hinges in 3D are currently prone to discontinuities due to the direction of a bone containing insufficient information to generate a consistently aligned orthonormal set of axes. To mitigate this problem, the library must be modified to store and maintain a model matrix per bone, or (ideally) to store bone directions as quaternions.

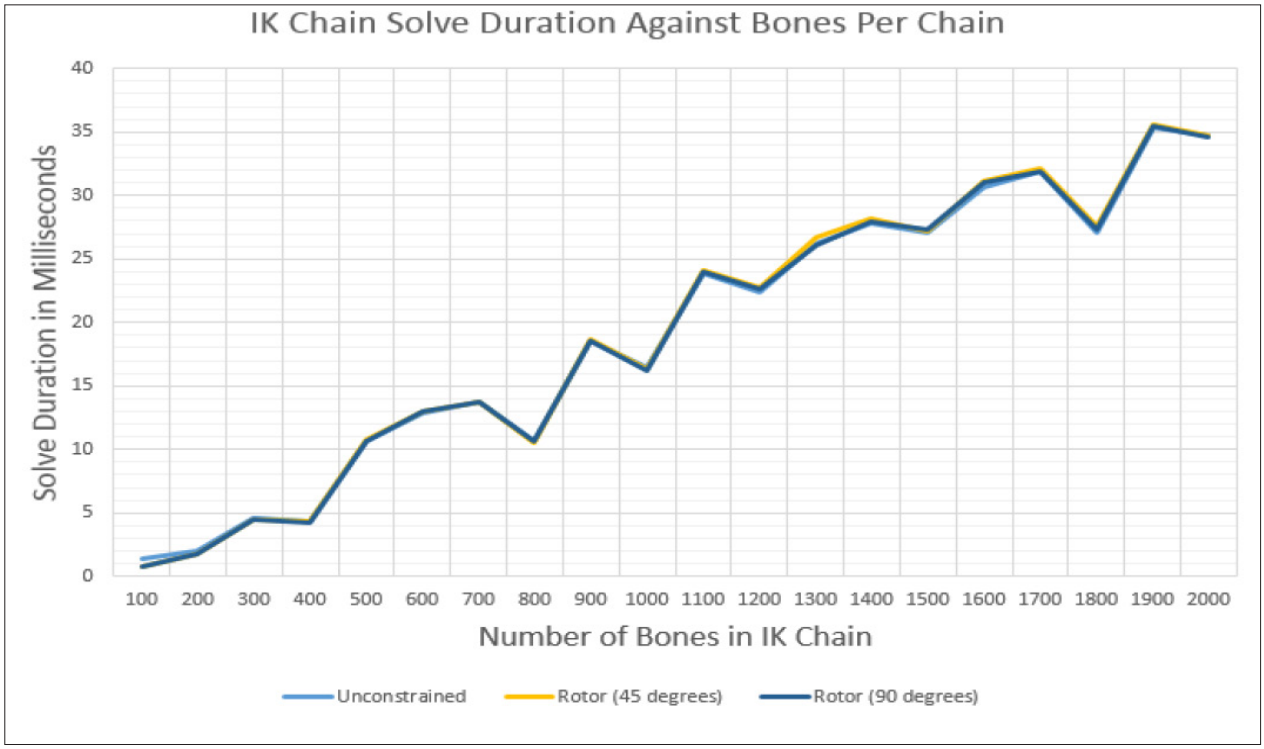

Figure 3: IK Chain solve duration in milliseconds vs. number of bones per chain as executed on an Intel i7-3610QM CPU. All chains for each sequence used the exact same target locations. As can be seen, constraints do not strongly affect performance. Chains with 1,000 bones can be solved in approximately 16ms (allowing for 60 updates per second). All chains were solved with the default acceptable solve distance of 0.1f and using the default of up to 20 iterations of the FABRIK algorithm per solve attempt.

(2) Availability

Operating system

Cross-platform Windows (Vista and above) and Linux (any modern distribution). For (optional) visualisation, OpenGL 3.3 with GLSL version 330 is required.

Programming language

Java 1.8

Additional system requirements

Any relatively modern PC can run the Caliko library.

In terms of storage requirements, the library as downloaded requires just over 2MB of storage. When built, which includes packaging the LWJGL3 library into the demonstration application, this increases to slightly under 7MB.

The demonstration application typically uses less than 75MB of system memory, and (optionally) uses a standard mouse and keyboard for input.

Dependencies

[Optional libraries for visualisation] LWJGL3 version: LWJGL 3.0.0b build 64, and OpenGL 3.3 with GLSL (OpenGL Shading Language) version 330 shaders.

List of contributors

Nil, other than the listed authors.

Software location

Archive (e.g. institutional repository, general repository) (required)

Name: Federation University Australia GitHub Account

Persistent identifier: DOI: 10.5281/zenodo.59285

Licence: MIT

Publisher: Alastair Lansley/Federation University Australia

Date published: 02/08/2016

Language

Git hosting. Maven packaging and built system. Java 1.8 classes. HTML technical documentation and PDF user documentation. Optional visualisation utilises OpenGL 3.3 and GLSL version 330 shaders.

(3) Reuse potential

Inverse kinematic techniques are typically used in controlling the limbs of robotic arms or skeletal movements. They are also commonly used in video games and 3D animation where they can, for example, allow for a character to reach for an item at a given location in a realistic fashion without the character first having to be aligned to a fixed starting point in relation to that item. Other potential usage scenarios include animation of multi-limbed insects or animals (with joint restrictions imposing limits on bone movement in order to constrain them to realistic

movement extents), or for animation of any provided human hand or body data where intermediary joint locations may be estimated from anatomical models.

As the FABRIK algorithm is typically considerably faster than other IK algorithms [1], it would be a good fit for IK in video games where processing occurs in real-time and therefore processing capacity is in high demand. Further, as the Caliko library provides a number of configurable settings for acceptance or resolving of a given chain configuration it may be easily tuned for high performance in a specific scenario (see the section on ***Implementation and Architecture*** for further details).

The Caliko source code is made freely available under the MIT license and may be extended upon as desired, or used as a reference implementation for transition into other programming languages suitable for inclusion in game or graphical engines such as Unity or Unreal Engine. An example of such a transition to another programming language can be seen in [7], where the 3D implementation has been translated into JavaScript.

No commercial support is available for the Caliko software, but issues may be reported via the standard GitHub issue tracking mechanism.

Acknowledgements

Many thanks to Andreas Aristidou and Joan Lasenby for the creation of the FABRIK algorithm, and to my PhD supervisors Peter Vamplew, Phil Smith and Cameron Foale for their support and guidance during the creation of the Caliko library.

Competing Interests

The authors declare that they have no competing interests.

References

1. **Aristidou, A** and **Lasenby, J** 2011 FABRIK: a fast, iterative solver for the inverse kinematics problem. *Graphical Models*, 73(5), pp. 243–260. DOI: http://dx.doi.org/10.1016/j.gmod.2011.05.003
2. **Leap Motion Sensor** Viewed 01 February 2016. https://www.leapmotion.com/product/desktop.
3. **The Caliko Inverse Kinematics Library** Viewed 05 February 2016. https://youtube/wEtp4P2ucYk.
4. **Apache Maven build management tool** Viewed 26 July 2016. https://maven.apache.org/.
5. **Buss, S R** 2004 Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, 17(1–19), p.16.
6. **Aristidou, A** and **Lasenby, J** 2009 *Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver*. University of Cambridge, Department of Engineering.
7. **FULLIK JavaScript version of the Caliko IK library** Viewed 02 August 2016. https://github.com/lo-th/fullik.

How to cite this article: Lansley, A, Vamplew, P, Smith, P and Foale, C 2016 Caliko: An Inverse Kinematics Software Library Implementation of the FABRIK Algorithm. *Journal of Open Research Software,* 4: e36, DOI: http://dx.doi.org/10.5334/jors.116

Published: 05 February 2016 **Accepted:** 18 August 2016 **Published:** 09 September 2016