Proceedings of the International Multiconference on
Computer Science and Information Technology pp. 933–937

ISBN 978-83-60810-14-9
ISSN 1896-7094

Solving IK problems for open chains using optimization methods

Krzysztof Zmorzynski
Warsaw University of Technology
Faculty of Mathematics and Information Science
Email: zmorzynskik@student.mini.pw.edu.pl

***Abstract*—Algorithms solving inverse problems for simple (open) kinematic chains are presented in this paper. Due to very high kinematics chains construction complexity, those algorithms should be as universal as possible. That is why optimization methods are used. They allow fast and accurate calculations for arbitrary kinematics chains constructions, permitting them to be used with success in robotics and similar domains.**

I. Introduction

THE problem of finding kinematic chain configuration for given position of its effector is called inverse kinematics (IK) problem. One of the major application of solving IK problem is robotics: having destination translation and orientation of the end effector, robot joints configuration should be found to achieve this translation and orientation. Robot mechanical constraints (for arms and joints) should be taken into consideration.

For some special cases, analytical and geometric methods for solving IK problems effectively exists [7], [9]. General explicit solution does not always have to exist, however. Also, as presented in [4], obtaining analytical solution might be very laborious and complex. These are the main drawbacks of analytical methods.

Methods based on Jacobian matrix and its inverse are widely described in literature [6], [8], [10]. When Jacobian matrix is not invertible, transposition or pseudoinverse Jacobian matrix is used instead [6], [8]. Computational cost of Jacobian matrix operations might be very high, though, not to mention stability problems in the neighborhoods of singularities.

In this paper optimization methods to solve IK problem are used. In contrast to methods mentioned above, they allow full automatization and perform all computations on a machine. They are also very universal, so can be used to solve arbitrary IK problem for open chains. Badler [1] used these methods to solve IK problem for more complicated chains containing many effectors, such as human skeleton. However, computation cost turned out to be to high to be used in real time, or even close to real time. Kim, Jang, Nam [2] used optimization methods to solve IK problem for binary manipulators (that is, each joint can take one of two possible positions).

More general manipulators are considered in this paper, with joints able to take any value from specified bounds. Supplied application allows user to modify chain construction, set effector's target translation and orientation and a real time visualisation of chain state during computations. It is a robust method to validate chain construction and check if desired target position can be achieved.

II. Chain representation

A. Chain configuration space

Open chains with each joint having single degree of freedom are considered in this paper. Whole chain configuration can be written as:

$$c = (q_1, q_2, \ldots, q_n) \in C \tag{1}$$

where C is chain configuration space, q_i is $i-th$ joint parameter and n is number of joints.

B. Joints

Two types of joints are specified: prismatic joints and revolute joints. Parameter value of each joint has to be between minimum and maximum values q_i^{min} and q_i^{max}. Joints are connected to each other along their local Z axes. It is also possible to set constant offset value along this axis (as presented in fig. 1). This is slightly simpler representation than in [5].

In the case of **prismatic** joint, parameter is additional (beside fixed offset mentioned above) length along local joint Z axis.

In the case of **revolute** joint, parameter is the rotation angle (in degrees) along specified local rotation axis. It is clear, that in order to model revolute joint with greater DOF, it is sufficient to create many revolute joints, with their fixed offsets set to 0, each allowing rotation about proper axis.

Beside parameter, each joint is represented also by its position, vector $g = (v, r)$, where g is element from space $L = \Re^3 \times S^3$. v is translation in world coordinates, r is rotation quaternion also in world coordinates. Joints translations and orientations are calculated in forward kinematics manner, starting from first (or root) joint. Formally, position of the effector is the map:

$$\begin{aligned} E : C &\rightarrow L \\ c \in C &\rightarrow E(c) \in L \end{aligned} \tag{2}$$

The use of quaternions instead of rotation matrices allows faster computations.

Effector's target translation and orientation is also represented by element e form space L.

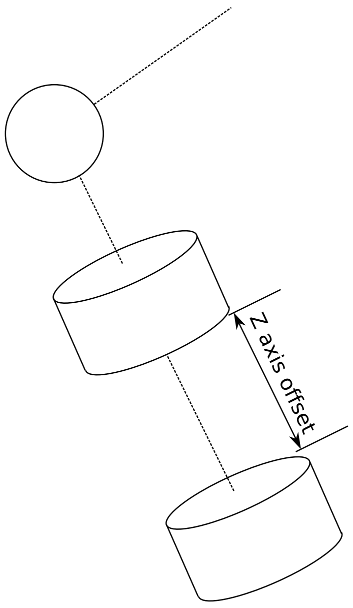


Fig. 1. Sample configuration. Dotted lines represent offsets along local Z axes.

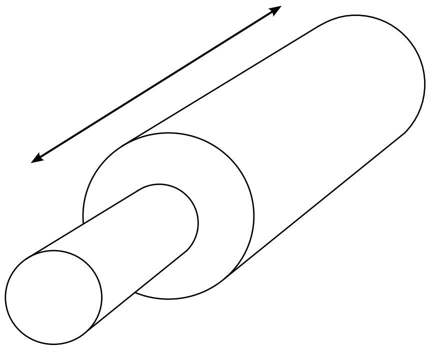


Fig. 2. Prismatic joint

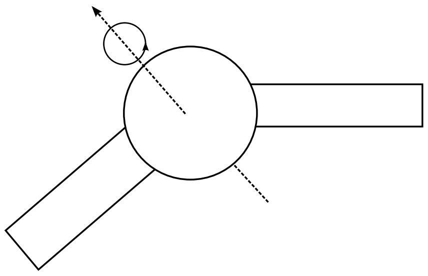


Fig. 3. Revolute joint. Dotted line is a rotation axis.

III. Optimization methods

A. Objective functions

In the case of solving IK problem, objective functions are generally distance (potential) functions from effector's current translation and orientation to effector's target translation and orientation. Written in a formal way, it is a function:

$$\begin{array}{rcl} d: L \times L & \rightarrow & \Re^+ \\ (x,y) \in L \times L & \rightarrow & d(x,y) \in \Re^+ \end{array} \tag{3}$$

where x and y are two elements from space L.

Following objective functions can be distinguished:

- translation only difference - Euclidean metric can be used:

$$d_{pos}(x,y) = \sqrt{(x_v - y_v) \cdot (x_v - y_v)} \tag{4}$$

  where $\cdot$ is dot product in $\Re^3$. Square root can be omitted in calculations.
- orientation only difference:

$$d_{rot}(x,y) = \sqrt{(x_r - y_r) \cdot (x_r - y_r)} \tag{5}$$

  where $\cdot$ is dot product of unit quaternions in S^3.
- translation and orientation difference at the same time:

$$d(x,y) = w_p d_{pos} + w_r d_{rot} \tag{6}$$

  where w_p and w_r are weights for translation and orientation functions, respectively.

B. Optimization methods and calculations

To minimise objective function, one can use gradient based methods, such as **conjugate gradient method**. It converges faster than steepest descent method since it takes conjugate (with respect to gradient) vector when choosing minimisation direction. Fletcher-Reeves formula for computing "direction factor" β can be used (see [3]). When β equals 1 it means that conjugate direction is no longer valid and should be set to 0 to make it steepest descent method.

Quasi-Newtonian methods, like **BFGS**, are even better than conjugate gradient. Although they require Hessian matrix computation, which may be expensive, it is also possible to use an approximation based on a function gradient value (described in [13]). The BFGS algorithm is presented in [12]. Its implementation available at [11] was used in the application.

One of the above functions, name it d (4-6), is chosen to be minimised. It takes current and target position of the effector. Recall that from eq. 2, its position is based on current chain configuration:

$$\begin{array}{rcl} c & \in & C \\ e = E(c) & \in & L \end{array} \tag{7}$$

where c is chain configuration.

It could also be written, that the following function f is being minimised:

$$\begin{array}{rcl} f: C & \rightarrow & \Re^+ \\ c \in C, y \in L & \rightarrow & f(c) = d(E(c), y) \in \Re^+ \end{array} \tag{8}$$

where c is current chain configuration, y is the effector's target position. The function f is minimised with respect to chain configuration and that means joints parameters. Joints parameters constraints should be taken into consideration while performing calculations (see next section).

C. Gradient computation

Gradient of the function f for conjugate gradient method is calculated using difference quotient (**two point**) method:

$$\bigtriangledown f(c) = (\frac{\partial f}{\partial q_1}(c), \ldots, \frac{\partial f}{\partial q_n}(c)) \tag{9}$$

where $c = (q_1, \ldots, q_n) \in C$ and $\frac{\partial f}{\partial q_i}$:

$$\frac{\partial f}{\partial q_i}(c) = \frac{f(q_1, \ldots, q_i + h, \ldots, q_n) - f(q_1, \ldots, q_i, \ldots, q_n)}{h}$$

where h is appropriate small value (see IV on how this value could be chosen). While calculating partial derivatives $\frac{\partial f}{\partial q_i}$, joints constrains q_i^{min} and q_i^{max} are taken into account. It is ensured that:

$$q_i^{min} \leq q_i + h \leq q_i^{max} \tag{10}$$

When value $q_i + h$ is out of the bounds, it is truncated to q_i^{min} or q_i^{max}.

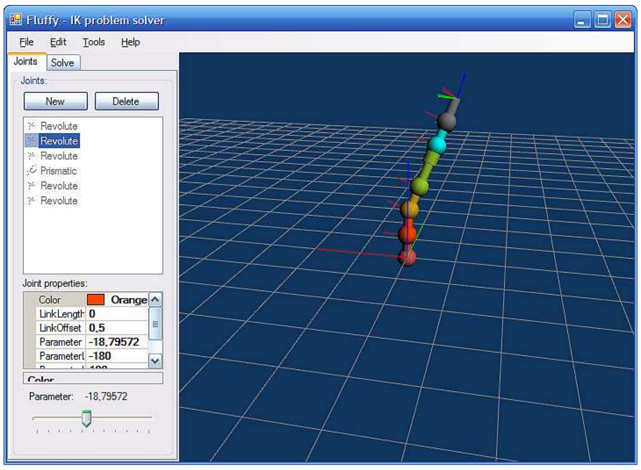
The use of BFGS method requires more precision in gradient computation. **Four point** method is used:

$$\begin{aligned} \frac{\partial f}{\partial q_i}(c) &= \frac{1}{12h}(f(q_1, \ldots, q_i - 2h, \ldots, q_n) \\ &- 8f(q_1, \ldots, q_i - h, \ldots, q_n) \\ &+ 8f(q_1, \ldots, q_i + h, \ldots, q_n) \\ &- f(q_1, \ldots, q_i + 2h, \ldots, q_n)) \end{aligned} \tag{11}$$

Thanks to definition of f like in eq. 8 it is possible to move from the original IK problem to the problem of multivariable objective function minimisation.

IV. Application

Created application allows user to interactively create and modify chain construction, save it to a file and load it later. Modification of joints parameters result in real time changes in chain visualisation. This applies to not only when user manipulates the parameters, but also while solving IK problem so it could be seen how the solution is being found. For given chain construction and end effector's target position, user chose which data types should be used in computations (see below), which objective function to minimise (see III-A) and which optimization method to use (see III-B). When calculations are finished, it is possible to view objective function value dependence on computation time and on iteration number.



Color Orange
LinkLength 0
LinkOffset 0,5
Parameter -18,79572
ParameterL -180

Parameter: -18,79572

Fig. 4. Application screenshot.

The application is written in `C# 2.0` language. Calculations are based on `generics` objects, so every floating point number type in this language: 32 bit `float`, 64 bit `double` and 128 bit `decimal` can be used in computations. This allows to easily tell how much each data type has impact on the efficiency and precision of calculations.

When calculating function gradient as stated in III-C, it is crucial to chose right value for h. As it turns out, it is not always when the smallest value performs best. "Batch" mode is available which allows user to compute with different values for h, so it could be estimated best for given chain construction and chosen data type.

By using the `.net 2.0` platform, it is possible to add new objective functions and optimisation methods solvers without need to recompile whole application. They are dynamically read at application startup from external `.net` dynamic link libraries (DLL) and added to internal database.

User can chose between `DirectX` or `OpenGL` graphics libraries used to render the scene. It is also possible to resign from rendering at all, increasing the same application performance.

V. Examples

Following examples were computed on a AMD Turion 64 X2 (two cores, each 1.6GHz) notebook, on Windows XP 32 bit with 1GB RAM and .net 2.0. Optimization algorithm was running on a separate thread, which also means on the other core than main application thread. Null renderer was selected to avoid unnecessary rendering overhead.

All calculations starts with all joints parameters set to 0 (fig. 5). Objective function 6 was used, with w_p set to 0.5 and w_r to 1.

Today's widely used robots have mechanical arms precision set at about 10^{-4} and 10^{-5} [m]. IK problems solvers should be able to find chain configuration to achieve one of those precision. In this paper both values were researched.

A. Example 1—relatively close to initial configuration

Chain with 6 joints was used, listed in order from the first (root) joint: first three revolute joints with rotation axes around X, Y, Z, respectively, then prismatic joint and then two additional revolute joints with rotation axes X and Z. All revolute joints has parameters constraints set to $(-90, 90)$ degrees, where prismatic joint has $(0, 2)$.

Effector's target position was set relatively close to chain's initial configuration. Figure 6 presents final calculated configuration.

All three data types were investigated, with h value (see III-C) set in batch mode. Tables I, II presents best times and values for each data type for 10^{-4} objective function precision,

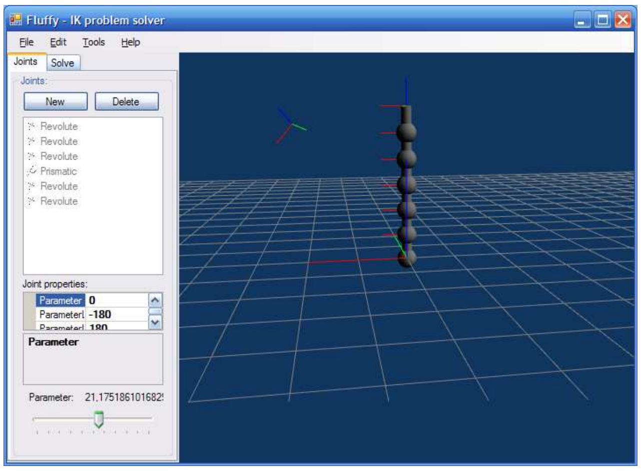


Fig. 5. Starting configuration.

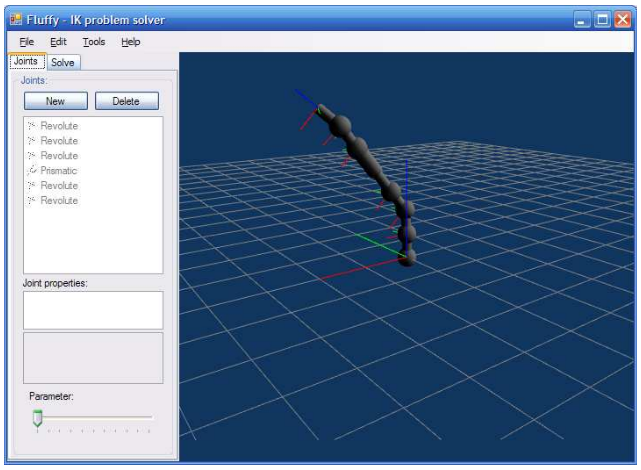


Fig. 6. Example 1 solution.

for conjugate gradient method and BFGS, respectively. Results for 10^{-5} are presented in tables III and IV.

TABLE I
EXAMPLE 1 RESULTS FOR 10^{-4} AND CONJUGATE GRADIENT METHOD

data type	h	iterations	time (ms)
`float`	10^{-5}	80	35
`double`	10^{-10}	75	65
`decimal`	10^{-12}	80	300

TABLE II
EXAMPLE 1 RESULTS FOR 10^{-4} AND BFGS METHOD

data type	h	iterations	time (ms)
`float`	10^{-2}	45	25
`double`	10^{-6}	45	45
`decimal`	10^{-2}	40	230

It can be seen from the tables I, II, III and IV, that `float` and `double` types performs best, due to their hardware support. `float` is fastest, probably because of smaller loading times from memory that `double`. Number of iterations for all

TABLE III
EXAMPLE 1 RESULTS FOR 10^{-5} AND CONJUGATE GRADIENT METHOD

data type	h	iterations	time (ms)
`float`	10^{-4}	105	48
`double`	10^{-10}	105	80
`decimal`	10^{-12}	90	320

TABLE IV
EXAMPLE 1 RESULTS FOR 10^{-5} AND BFGS METHOD

data type	h	iterations	time (ms)
`float`	10^{-2}	50	30
`double`	10^{-10}	40	45
`decimal`	10^{-2}	45	250

types are similar. `decimal` type has significant calculation impact—nearly 10 times slower than `float`, mainly because of its software `.net` emulation.

Clearly, BFGS method is faster by about 50% than conjugate gradient method. It also needs half of the iterations to find the solution.

Figure 7 presents rough overview of how convergence was achieved in the matter of iterations (left plot) and time (right plot).

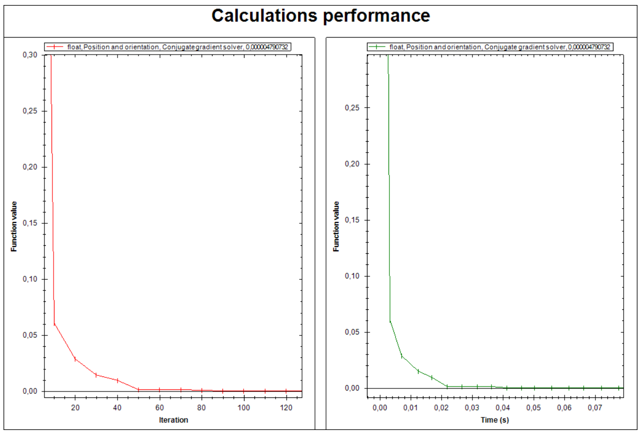


Fig. 7. Example 1 convergence overview.

B. Example 2

This time effector's target position was farther than in previous example (as in fig. 8), so computation time was expected to be longer. Also, to ensure that effector reaches target position, additional revolute joint was added at the end of the chain, and each revolute joint has constraints set to $(-180, 180)$ degrees. All revolute joints, except the first and the last one, have fixed offset set to 0.

Results are presented in tables V and VII.

TABLE V
EXAMPLE 2 RESULTS FOR 10^{-4} AND CONJUGATE GRADIENT METHOD

data type	h	iterations	time (ms)
`float`	10^{-3}	105	50
`double`	10^{-6}	90	65
`decimal`	10^{-5} to 10^{-12}	~110	~400

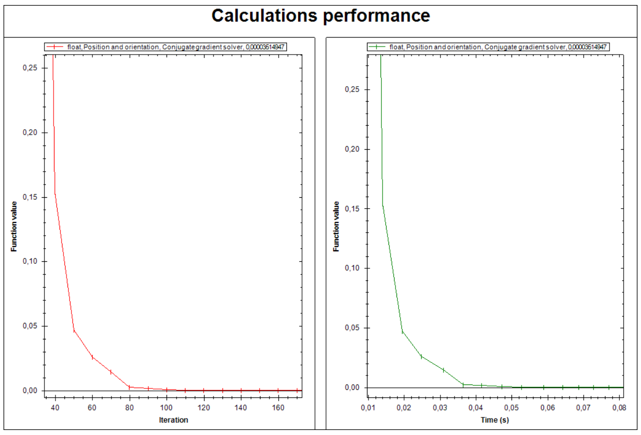


Fig. 9. Example 2 convergence overview.

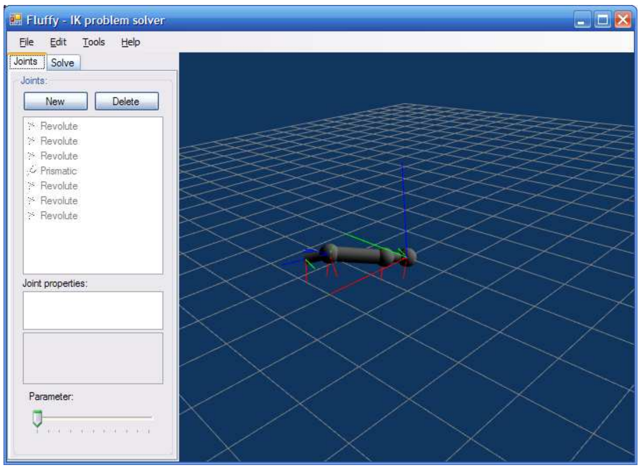


Fig. 8. Example 2 solution.

TABLE VI
EXAMPLE 2 RESULTS FOR 10^{-4} AND BFGS METHOD

data type	h	iterations	time (ms)
`float`	10^{-2}	40	25
`double`	10^{-1}	35	45
`decimal`	10^{-1} to 10^{-12}	˜35 – 40	˜250

TABLE VII
EXAMPLE 2 RESULTS FOR 10^{-5} AND CONJUGATE GRADIENT METHOD

data type	h	iterations	time (ms)
`float`	10^{-3}	120	58
`double`	10^{-6}	100	76
`decimal`	10^{-5} to 10^{-12}	˜120	˜500

TABLE VIII
EXAMPLE 2 RESULTS FOR 10^{-5} AND BFGS METHOD

data type	h	iterations	time (ms)
`float`	10^{-3}	45	30
`double`	10^{-1}	40	50
`decimal`	10^{-1} to 10^{-12}	˜35 – 40	˜300

This time, calculation times excludes `decimal` type from any practical usage. For conjugate gradient method, two other types performs slightly worst than in the previous example, but still in the almost real-time. For BFGS method, there are almost no differences between both examples. For all data types, iteration count is quite similar to each other.

Figure 9 shows how convergence was achieved.

VI. CONCLUSIONS AND FUTURE WORK

It is clear from above examples, that optimization methods can be widely used to solve IK problems in the real time, even on commonly available customers computers. This leaves no need of use dedicated and expensive workstations, as it has been before.

Applying commonly used and available methods (such as conjugate gradient and BFGS) seems to work well in most cases. Their adaptation to multithreaded or multiprocessor environments may perform even better.

REFERENCES

[1] J. Zhao and N. I. Badler, *Inverse kinematics positioning using nonlinear programming for highly articulated figures*, ACM Transactions on Graphics, Vol. 113, No. 4, October 1994, pages 313–336.
[2] Y. Y. Kim, G. W. Jang and S. J. Nam *Inverse kinematics of binary manipulators by the optimization method in continous variable space*, IEEE International Conference on Intelligent Robots and Systems, September 28 – October 2, 2004, Sendai, Japan.
[3] J. R. Shewchuk *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*, School of Computer Science, Carnegie Mellon University, August 4, 1994.
[4] J. Angeles *Fundamentals of Robotic Mechanical Systems: Theory, Methods and Algorithms*, Springer, 2nd edition, 2003.
[5] L. T. Wang and B. Ravani *Recursive Computations of Kinematic and Dynamic Equations For Mechanical Manipulators*, IEEE Journal of Robotics and Automation, VOL. RA-1, NO. 3, September 1985.
[6] S. R. Buss, *Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Dumped Least Squares methods*, University of California, San Diego, April 2004.
[7] Kang Teresa Ge *Solving Inverse Kinematics Constraint Problems for Highly Articulated Models*, Masters of Science thesis, University of Waterloo, 2000
[8] D. Park *Inverse Kinematics*, Computer Graphics, Department of Computer Science, University of Buenos Aires, Argentina.
[9] X. Wu et al. *A 12-DOF Analytic Inverse Kinematics Solver for Human Motion Control*, Journal of Information and Computational Science 1: 1 2004, pages 137–141.
[10] W. Stadler, P. Eberhard *Jacobian motion and its derivatives*, Mechatronics 11, 2001, pages 563–593.
[11] S. Bochkanov, V. Bystritsky *AlgLib - BFGS-B - http://www.alglib.net/optimization/lbfgsb.php*, 1999–2008.
[12] R. H. Byrd, P. Lu and J. Nocedal. *A Limited Memory Algorithm for Bound Constrained Optimization*, SIAM Journal on Scientific and Statistical Computing, 1995, 16, 5, pp. 1190–1208.
[13] J. Nocedal, S. J. Wright *Numerical Optimization*, Springer-Verlag, 1999.