A Study of Inverse Kinematics Algorithm and Its System Architecture Implementation

Yuxin Yang, Tao Song

Abstract—Inverse Kinematics is a compute-intensive tasks in robotics. It is hard to calculate Inverse Kinematics when robots degree grow up for embed processor. We introduced an parallel algorithm Rand-IK to improve the calculation speed. It can run on GPU, FPGA or ASICs.

Index Terms—Robots, Inverse Kinematics, Jacobian, Transpose, Algorithm, Architecture

1 Introduction

With the development of robotics and 3D animations, more and more complex real or virtual robots appear. Kinematics, the base of robotics, is becoming more and more important. When deal with a high degree robots in real or virtual environment, we usually get stuck because the inverse kinematics is hard to calculate. Many algorithm has been proposed to solve these problem. Their three types of IK algorithm: analytical method, iterative method and parametric method. Analytical method is very fast, has high precision, with strong stability, but it has many limit to robots itself. iterative method has high precision, strong universality, but the speed is slow, stability is weak. Parametric method is fast, stable, but not precision.

2 Kinematics and Jacobians

Kinematics can describe the behavior of robots. In most cases, we deal with the kinematics under the model of tree. The nodes of the tree represent the joints of robot, and the edges of the tree represent the links of the robot. The joints have many types, mainly including revolute joints and prismatic joints. Revolute joint allows one link rotating around adjacent one, alone a typical axis. Prismatic joint allow one link translating alone a typical axis. There are two goals of Kinematics. One is giving the state of the joints, then calculate the end effectors' pose of the robot, which is called forward kinematics. Another one is giving the end effectors' pose, then calculate the possible state of the joints, which is called inverse kinematics. We can denote the state of the joints by $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_n)^{\mathsf{T}}$, , which represent the rotation angel of the revolute joints or the translation distance of the prismatic joints. The end effectors' pose is denoted by $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)^{\mathsf{T}}$, where $m = 6k$, k is the number of the end effector. For the pose of one end effector, 3 argument represent the position in 3D space, another 3 argument represent the axis direction of the end effector. Then we can generally represent the forward kinematics as

$$\boldsymbol{s} = f(\boldsymbol{\theta}). \tag{1}$$

And inverse kinematics can be represented as:

$$\boldsymbol{\theta} = f^{-1}(\boldsymbol{s}). \tag{2}$$

In general, (1) is simple, but (2) is hard to resolve. In some cases, (2) is unsolvable, and if $m > n$, (2) may have multi solutions. Even deal with small(such as 6) degree robots, (2) may be no closed form solution.

Jacobian method is widely used when deal with (2). Jacobian matrix $\boldsymbol{J}$ is a linear approximation of (1) at point $\boldsymbol{\theta}$:

$$\boldsymbol{J}(\boldsymbol{\theta}) = (\frac{\partial x_i}{\partial \theta_j})_{i,j}. \tag{3}$$

If we change $\boldsymbol{\theta}$ by $\Delta\boldsymbol{\theta}$, we will have:

$$\Delta\boldsymbol{x} = \boldsymbol{J}(\boldsymbol{\theta})\Delta\boldsymbol{\theta}. \tag{4}$$

Then $\Delta\boldsymbol{\theta}$ can be estimated as:

$$\Delta\boldsymbol{\theta} = \boldsymbol{J}^{-1}(\boldsymbol{\theta})\vec{e}. \tag{5}$$

Where $\vec{e} = \Delta\boldsymbol{x} = \boldsymbol{x}_t - \boldsymbol{x}_s$, $\boldsymbol{x}_s$ is current pose and $\boldsymbol{x}_t$ is target pose. Then we can add $\Delta\boldsymbol{\theta}$ to $\boldsymbol{\theta}$. This is one iteration:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}. \tag{6}$$

After several iterations, we will get an approximate solution of (2).

Unfortunately, in most cases, Jacobian matrix $\boldsymbol{J}$ is not square, so it is invertible. Even $\boldsymbol{J}$ is invertible, the solution of (5) will become bad when $\boldsymbol{J}$ is near singular.

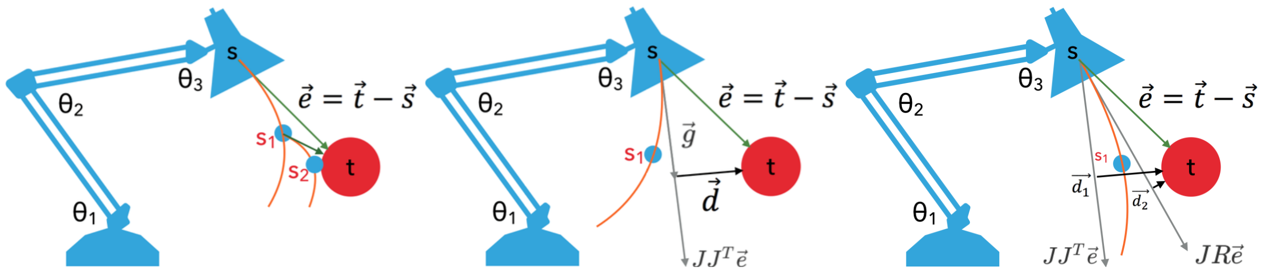
To solve this problem, Jacobian pseudoinverse method and Jacobian transpose method was proposed.

$$\Delta\boldsymbol{\theta} = \boldsymbol{J}^{\mathsf{T}}(\boldsymbol{J}\boldsymbol{J}^{\mathsf{T}})^{-1}\vec{e}. \tag{7}$$

$$\Delta\boldsymbol{\theta} = \alpha\boldsymbol{J}^{\mathsf{T}}\vec{e}. \tag{8}$$

Jacobian pseudoinverse method use (7) to deal with unsquared matrix $\boldsymbol{J}$ and find a direction $\boldsymbol{J}\Delta\boldsymbol{\theta} = \boldsymbol{J}\boldsymbol{J}^{\mathsf{T}}(\boldsymbol{J}\boldsymbol{J}^{\mathsf{T}})^{-1}$ which is almost the same with $\vec{e}$. But when $\boldsymbol{J}$ is singular, the performance of pseudoinverse method is bad.

• *Yuxin Yang and Tao Song are with 1.Center for Intelligent Computing Systems, State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. 2.University of Chinese Academy of Sciences, Beijing, China.*
E-mail: {yangyuxin2018,songtao19g}@ict.ac.cn



(a) Jacobian pseudoinverse method (b) Jacobian transpose method (c) Rand-IK method

Fig. 1: Illustration of different IK method.

Jacobian transpose method use $\boldsymbol{J}^{\mathsf{T}}$ instead of $\boldsymbol{J}^{-1}$, and apply a scalar α to calculate $\Delta\boldsymbol{\theta}$. It can be prove that the direction $\boldsymbol{J}\boldsymbol{J}^{\mathsf{T}}\vec{e}$ is a right direction, which means if we change $\boldsymbol{\theta}$ by $\Delta\boldsymbol{\theta}$, the distance $\|\vec{d}\|_2$ between the end effectors' pose and the target pose will become smaller. Just because $\langle \boldsymbol{J}\boldsymbol{J}^{\mathsf{T}}\vec{e},\vec{e}\rangle = \|\boldsymbol{J}^{\mathsf{T}}\vec{e}\|_2 >= 0$. And scalar α is an estimated value, to minimize the distance $\|\vec{d}\|_2$, alone the direction $\boldsymbol{J}\boldsymbol{J}^{\mathsf{T}}\vec{e}$. If $f(\boldsymbol{\theta})$ is linear, then $\alpha = \frac{\langle \boldsymbol{J}\boldsymbol{J}^{\mathsf{T}}\vec{e},\vec{e}\rangle}{\langle \boldsymbol{J}\boldsymbol{J}^{\mathsf{T}}\vec{e},\boldsymbol{J}\boldsymbol{J}^{\mathsf{T}}\vec{e}\rangle}$.

Fig. 1a is Jacobian pseudoinverse method. The linearized direction is the same as $\vec{e}$. The real trajectory is the curve diverge from the direction. After one step, S move to S_1, then move to S_2 the next step. Fig. 1b is Jacobian transpose method. The linearized direction $\boldsymbol{J}\boldsymbol{J}^{\mathsf{T}}\vec{e}$ let $\|\vec{d}\|_2$ become smaller, and $\vec{g} = \alpha\boldsymbol{J}\boldsymbol{J}^{\mathsf{T}}\vec{e}$ which minimize $\|\vec{d}\|_2$. The real trajectory may diverge far away from the direction.

Jacobian transpose method can avoid matrix inverse operator, which is hard to compute. It also performs well in singular situation. But it's step direction is inaccuracy, the algorithm need many more steps to converge, which consume a lot of time.

3 Rand-IK Algorithm

Rand-IK Algorithm is an Inverse Kinematics algorithm design for hardware implementation. It takes advantage of both Jacobian pseudoinverse method and Jacobian transpose method. The core idea is using the parallel computation ability to find a better direction, and a best scalar α under this direction.

Rand-IK algorithm using the direction $\boldsymbol{J}\boldsymbol{R}\vec{e}$, instead of $\boldsymbol{J}\boldsymbol{J}^{\mathsf{T}}\vec{e}$ in Jacobian transpose method. As in Fig. 1c, the iterative direction is closer to the target pose than one in Jacobian transpose method. But it is still not the same as $\vec{e}$.

$$\Delta\boldsymbol{\theta} = \alpha\boldsymbol{R}\vec{e}. \tag{9}$$

The matrix $\boldsymbol{R}$ has same size with $\boldsymbol{J}^{\mathsf{T}}$, and the element in it has a randomized bias around the element of $\boldsymbol{J}^{\mathsf{T}}$. We can parallel compute multiple matrix $\boldsymbol{R}$ and chose one with smallest $\|\vec{d}\|_2$. Then We have to determine the scalar α. The direction $\boldsymbol{J}\boldsymbol{R}\vec{e}$ is the linearized direction, the real trajectory of end effector alone direction $\boldsymbol{J}\boldsymbol{R}\vec{e}$ need to use forward kinematics algorithm to compute. We have to use multiple scalar α to compute the real end effectors' pose, and chose a best one. Now we get a direction $\boldsymbol{J}\boldsymbol{R}\vec{e}$ and a scalar α, then we can update $\boldsymbol{\theta}$ step by step.

The whole algorithm is shown blow.

Algorithm 1 Rand-IK

```
Input: Config $\boldsymbol{C}$, Current joints state $\boldsymbol{\theta}$, Target pose $\boldsymbol{x}_t$
Output: Target joints state $\boldsymbol{\theta}$
 1: if did not init then
 2:   init($\boldsymbol{C}$)
 3: end if
 4: for $i \leftarrow 0 : maxSteps$ do
 5:   $\boldsymbol{x}_s \leftarrow f(\boldsymbol{\theta})$
 6:   $\vec{e} \leftarrow \boldsymbol{x}_s - \boldsymbol{x}_t$
 7:   if $\|\vec{e}\|_2 < eps$ then
 8:     return $\boldsymbol{\theta}$
 9:   end if
10:   $\boldsymbol{J} \leftarrow \boldsymbol{J}(\boldsymbol{\theta})$
11:   $k \leftarrow 0$
12:   $d_{min} \leftarrow inf$
13:   for $j \leftarrow 0 : maxRandomTimes + 1$ do
14:     generate $\boldsymbol{R}_j$ $\{\boldsymbol{R}_0 = \boldsymbol{J}^{\mathsf{T}}\}$
15:     $\alpha_{j0} \leftarrow \frac{\langle \boldsymbol{J}\boldsymbol{R}_j\vec{e},\vec{e}\rangle}{\langle \boldsymbol{J}\boldsymbol{R}_j\vec{e},\boldsymbol{J}\boldsymbol{R}_j\vec{e}\rangle}$
16:     $\vec{d} \leftarrow \vec{e} - \alpha_{j0}\boldsymbol{J}\boldsymbol{R}_j\vec{e}$
17:     if $\|\vec{d}\|_2 < d_{min}$ then
18:       $d_{min} \leftarrow \|\vec{d}\|_2$
19:       $k \leftarrow j$
20:     end if
21:   end for
22:   $t \leftarrow 0$
23:   $e_{min} \leftarrow inf$
24:   for $j \leftarrow 1 : maxSSUNumber + 1$ do
25:     $\alpha_{kj} \leftarrow \alpha_{k0} * maxRange * \frac{j}{maxSSUNumber}$
26:     $\boldsymbol{x}_j \leftarrow f(\boldsymbol{\theta} + \alpha_{kj}\boldsymbol{R}_k\vec{e})$
27:     $\vec{e} \leftarrow \boldsymbol{x}_j - \boldsymbol{x}_t$
28:     if $\|\vec{e}\|_2 < e_{min}$ then
29:       $e_{min} \leftarrow \|\vec{e}\|_2$
30:       $t \leftarrow j$
31:     end if
32:   end for
33: end for
34: return $\boldsymbol{\theta} + \alpha_{kt}\boldsymbol{R}_k\vec{e}$
```

4 Framework

The Rand-IK algorithm is designed for hardware with parallel computation ability, such as GPU, FPGA or ASIC. GPU is easy to implementation, but the speed of GPU's core is relative slow, and it is hard for GPU to utilize the pipeline optimization for the loops. ASIC is fast, but it is too hard to implementation. So FPGA is a good choice. Fig. 2 show the

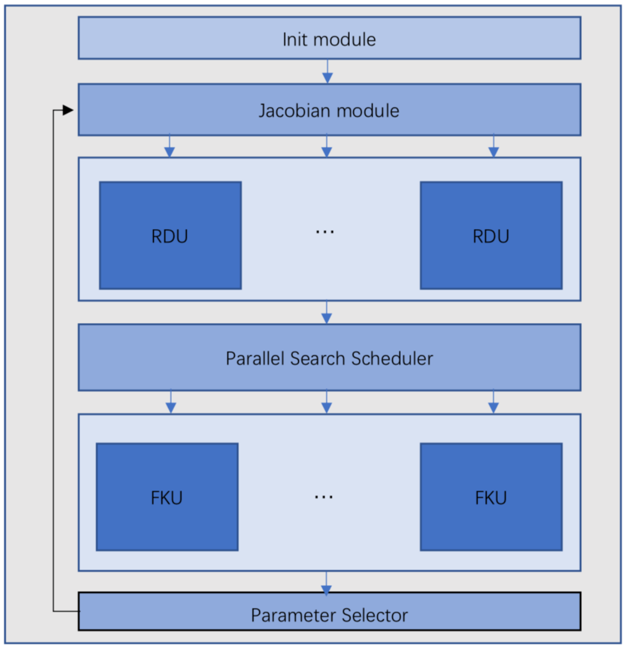


Fig. 2: Framework of Rand-IK

framework of Rand-IK.

4.1 Init module

To begin with an inverse kinematics algorithm, the accelerator must know the configuration of the robot in advance. So the accelerator have to judge if the initialization procedure has done. If not, read the configuration information and set the relative memory in the accelerator. This module do not affect the speed of the accelerator.

4.2 Jacobian module

The Jacobian module can calculate Jacobian Matrix and end effectors' current pose (which could judge whether answer is found, if found then return). It need to scan the tree model twice, operate a series of matrix multiplication. To simplify the implementation of FPGA or ASIC, we need to fix the degree of the robots, and set only one end effector. So the tree structure can become a chain. Then it is easy to unroll the loops, and calculate the series of matrix multiplication using merge algorithm, which can take advantage of parallel resources.

4.3 Random Direction Unit

Each RDU need to generate a randomized matrix $\boldsymbol{R}$, calculate the scalar value α and estimated distance $||\vec{d}||_2$ for several times in pipeline. The randomized matrix $\boldsymbol{R}$ do not need to store, we just need to store the best result of $\boldsymbol{R}\vec{e}$. To generate a randomized matrix, we just need pre-store a random matrix $\boldsymbol{R}_0$, the same size as $\boldsymbol{R}$, and use different index and offset to identify different $\boldsymbol{R}$.

4.4 Parallel Search Scheduler

The Parallel Search Scheduler find the best result of RDU produced. Then calculate the candidate joints state, transfer it to different FKU.

4.5 Forward Kinematics Unit

Each FKU can procedure several input. The core operator in FKU is just the 4x4 matrix multiplication. We can calculate these series of matrix multiplication with merge algorithm, and in pipeline.

4.6 Parameter Selector

Find the best result, then update the joints state, begin next step.

5 Experiment

We implement the Rand-IK algorithm in CPU and FPGA (which can be obtained at https://github.com/CICS-ICT/ik-acceleration), comparing with other algorithm. Our CPU is Cure i7-8750, FPGA implementation is synthesized by Vivado HLS, and evaluated by the synthesis report.

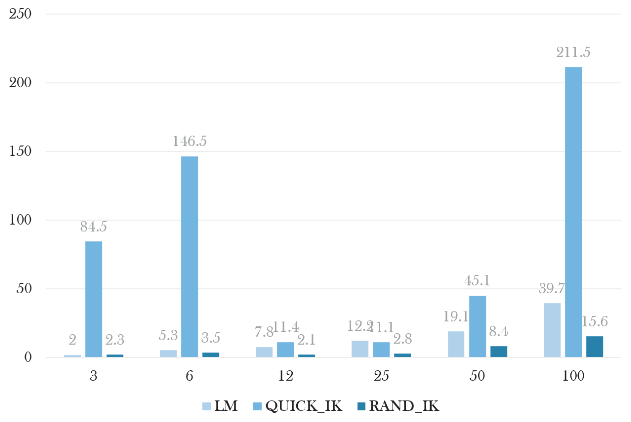


Fig. 3: Performance comparison(ms)

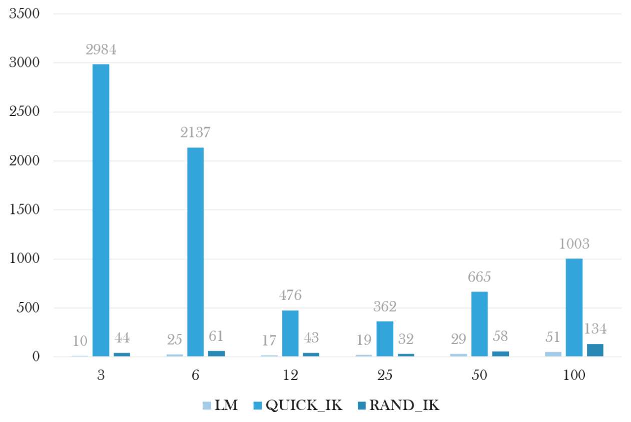


Fig. 4: Steps comparison

5.1 Speed

The time estimation is shown as below. LM is a very fast algorithm in KDL tools running on CPU. Quick-IK is an algorithm running on ASIC(whose frequency is 1G Hz). In order to compare the performance of the algorithm fairly, we running it on CPU and then evaluate its highest possible performance. The highest possible performance of Rand-IK is also evaluated by CPU realization. As Fig. 3 shows, Rand-IK is the fastest one. The tradition Jacobian transpose method is too slow, so we didn't run it. The horizontal axis represents degrees of freedom. Our FPGA implementation is running in 100M Hz. So it will be slightly slower than IKAcc(1G Hz) in Dadu.

5.2 Steps

As Fig. 4 shows, LM algorithm has the smallest iteration steps, because it is one kind of Jacobian pseudoinverse method, has the most accurate direction each step. Our method Rand-IK is better than tradition Jacobian transpose method and Quick-IK Algorithm.

6 Conclusion

The algorithm Rand-IK is indeed better than Quick-IK, but the frequency of FPGA is too low, resulting in the final performance evaluation is not as good as Dadu implemented by ASIC. At the same time, the rand-ik implemented by FPGA consumes a lot of hardware resources. Compared with the LM algorithm implemented by CPU, it only improves a little performance, which is not very cost-effective.

References

[1] S. Lian et al., "Dadu: Accelerating Inverse Kinematics for high-DOF robots," in 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), Jun. 2017, pp. 1–6, doi: 10.1145/3061639.3062223.

[2] Z. Jiang, Y. Dai, J. Zhang, and S. He, "Kinematics calculation of minimally invasive surgical robot based on FPGA," in 2017 IEEE International Conference on Robotics and Biomimetics (ROBIO), Dec. 2017, pp. 1726–1730, doi: 10.1109/ROBIO.2017.8324667.

[3] S.- Park and J.- Oh, "Inverse kinematics for robot manipulators based on incremental unit computation method," in Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93), Jul. 1993, vol. 3, pp. 1622–1626 vol.3, doi: 10.1109/IROS.1993.583855.

[4] M. Kameyama, T. Matsumoto, H. Egami, and T. Higuchi, "Implementation of a high performance LSI for inverse kinematics computation," in 1989 International Conference on Robotics and Automation Proceedings, May 1989, pp. 757–762 vol.2, doi: 10.1109/ROBOT.1989.100075.

[5] Y.-S. Kung, B. T. H. Linh, M.-K. Wu, F.-C. Lee, and W.-C. Chen, "FPGA-Realization of Inverse Kinematics Control IP for Articulated and SCARA Robot," in Design and Computation of Modern Engineering Materials, A. Öchsner and H. Altenbach, Eds. Cham: Springer International Publishing, 2014, pp. 205–213.

[6] F. Schwiegelshohn, F. Kästner, and M. Hübner, "FPGA design of numerical methods for the robotic motion control task exploiting high-level synthesis," in 2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE), Nov. 2016, pp. 1–5, doi: 10.1109/ICSEE.2016.7806074.

[7] K. Gac, G. Karpiel, and M. Petko, "FPGA based hardware accelerator for calculations of the parallel robot inverse kinematics," in Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012), Sep. 2012, pp. 1–4, doi: 10.1109/ETFA.2012.6489717.

[8] M. Wu, Y. Kung, Y. Huang, and T. Jung, "Fixed-point computation of robot kinematics in FPGA," in 2014 International Conference on Advanced Robotics and Intelligent Systems (ARIS), Jun. 2014, pp. 35–40, doi: 10.1109/ARIS.2014.6871523.

[9] W. Chen, C. Chen, F. Lee, and Y. Kung, "Digital hardware implementation of the forward/inverse kinematics for a SCARA robot manipulator," in 2018 IEEE International Conference on Applied System Invention (ICASI), Apr. 2018, pp. 54–57, doi: 10.1109/ICASI.2018.8394315.

[10] Y. Kung, M. Wu, B. T. H. Linh, T. Jung, S. Lee, and W. Chen, "Design of inverse kinematics IP for a six-axis articulated manipulator," in 2013 CACS International Automatic Control Conference (CACS), Dec. 2013, pp. 300–305, doi: 10.1109/CACS.2013.6734150.

[11] M. Petko, K. Gac, G. Karpiel, and G. Góra, "Acceleration of parallel robot kinematic calculations in FPGA," in 2013 IEEE International Conference on Industrial Technology (ICIT), Feb. 2013, pp. 34–39, doi: 10.1109/ICIT.2013.6505644.