Using Particle Filters to Solve the Problem of Symmetric Multiple Solutions in Robot Inverse Kinematics

Chien-Lin Chiang[1], Chang-Chen Hsieh[2], Yi-Yuan Chiang[2], I-Long Lin[1]*

[1] Department of Computer Science and Engineering, Tatung University, Taiwan
[2] Department of Computer Science and Information Engineering, Vanung University, Taiwan
{f2267505, superlongtw}@gmail.com, yychiang@mail.vnu.edu.tw, cyberpaul@ttu.edu.tw

Abstract

Encountering multiple solutions is the most severe problem in inverse kinematics of robot arms. Multiple solutions usually appear in symmetrical forms; it means that the joint angles vary greatly between two symmetrical postures. During the operation of the robot arm, the path through any joint space should be smooth. Otherwise, there will inevitably be situations where the joint angle jumps significantly between two adjacent timesteps. In this paper, we propose a novel method to solve robotics inverse kinematics for manipulators. This method utilizes particle filters to track the possible postures of planar robot arms given the positions of the end effectors. In this way, the poses of the robot arms are modeled as particles within these filters. The particle filter algorithm is an iterative process, which tracks the angles of joints by averaging the particles and regenerates populations so that particles can converge to the arm poses. In addition, particles can remember the previous values after regeneration, so that the inverse poses do not follow a non-differentiable path in the joint spaces. To verify the effectiveness of the proposed method, we implemented a simulator and tested the performance of the particle filters in a nonlinear end terminal path.

Keywords: Manipulators, Inverse kinematics, Multiple solutions, Symmetry

1 Introduction

In the era of Industry 4.0, robotic arms have become essential equipment in many industries. Their applications are also very extensive, such as automobile production, electronic assembly, material handling, machining, medical, and many other fields [1-4]. In inverse kinematics for robotic arms, it is common to encounter the issue of multiple solutions, which often results in interference from multiple symmetric poses. When faced with multiple solutions, the robotic arm must choose one set of solutions to determine its posture. While symmetric solutions are often reasonable choices for robotic arm postures, an inappropriate selection during motion can cause rapid changes between postures, resulting in severe vibrations in the mechanism. This study aims to explore the posture selection of robotic arms to avoid interference from symmetric multiple solutions. We have studied some practical solutions, where geometric methods can be used to calculate the kinematic and mechanical characteristics of robotic arms [5-8]. When solving the inverse kinematics problem of robotic arms, modeling and simulation of robotic arms in joint space can be used to describe joint restrictions and motion range, avoid collision, calculate the optimal motion path, and analyze and control the posture and motion of robotic arms. Algebraic methods can also be used to control the motion and posture of the arm [9-13]. Using the kinematic model and inverse kinematic algorithm, algebraic methods can be used to calculate the joint angles of the robotic arm to achieve the required motion path of the robotic arm and solve the position and posture of the robotic arm in space. To prevent interference from symmetric multiple solutions, we use non-parametric Bayesian filters (also known as particle filters), which are based on the Monte Carlo method instead of traditional algebraic or geometric methods of inverse kinematics. The particle filters are designed with a large number of random sampling points and importance weights, making the sampling points converge to the optimal solution during iterative computations. In this paper, we introduce a novel approach for solving inverse kinematics problems for robotic manipulators. Our method employs particle filters to trace the feasible postures of planar robotic arms based on the given end effector positions. The particle filters model the poses of the robot arms as particles and employ an iterative algorithm that averages the particles' joint angles and regenerates populations to converge to the desired arm poses. Furthermore, particles remember their previous values after regeneration, ensuring the inverse poses have a differentiable path in the joint spaces. We validate the effectiveness of our approach by implementing a simulator and evaluating the performance of the particle filters in a nonlinear end terminal path.

The rest of this paper is organized as follows: Section 2 reviews typical methods of inverse kinematics, including the geometrical method, algebraic method, and neural network approach. Section 3 describes the main contribution of this paper, the particle filter approach to inverse kinematics. Experimental results are presented in Section 4 followed by Conclusions in Section 5.

*Corresponding Author: Yi-Yuan Chiang; E-mail: yychiang@mail.vnu.edu.tw
DOI: https://doi.org/10.70003/160792642024072504006

2 Inverse Kinematics

In the study of inverse kinematics, the end point of a robot is given. Then, the angle of each joint should be determined. The following subsections show the typical methods.

2.1 Geometric Method

The use of the geometric method can provide visualized solutions [5-7]. Zonggao Mu et al. [14] showed a segmented geometric method for solving the configuration planning problems. It can be applied to different types of manipulators, such as parallel-jointed, orthogonal-jointed, or universal-jointed structures. Meanwhile, Samer Yahya et al. [15] presented a method for determining the configuration of the end effector on a smooth path composed of points that are close enough.

The geometric method can not only be used to solve the inverse kinematics problem of 2D planar robots but also 3D robots. For 3D cases, the 3D geometric shape can be separated into 2D plane geometric shapes. The geometric method as shown in Figure 1. The endpoint of Link 2 is (x, y). Use the cosine law to find θ_2. The angle can be drawn as two solutions: the green triangle and the orange triangle. For example, using the geometric method, the space can be divided into planar geometry. Since l_1 and l_2 are the lengths of Link1 and Link2, respectively, we can use the cosine law to find the angle $c2$. Therefore, we have the following equation:

$$x^2 + y^2 = l_1^2 + l_2^2 - 2l_1 l_2 \cos(\pi - \theta_2) \ . \tag{1}$$

$$\cos(\theta_2) = \frac{x^2 + y^2 - l_1^2 + l_2^2}{2l_1 l_2} \ . \tag{2}$$

Then, $\theta 2$ along with $\theta 1$ and $\theta 3$ could be solved by inverse trigonometric functions.

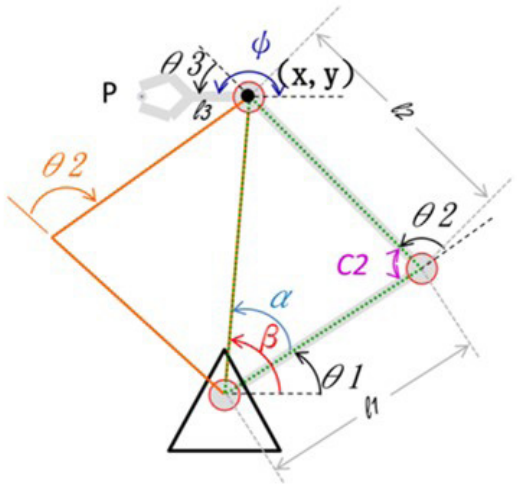

Figure 1. Symmetric multiple solutions in inverse kinematics of an RRR type robot arm

While the geometric method can solve the inverse kinematics problem of the mechanical arm and provide various useful methods, we also see some limitations, such as being used for complex mechanical structures and high-dimensional kinematic problems, which require a large amount of computation time and resources. In environments that demand high precision, if sufficient mathematical models are not provided to describe the kinematic and mechanical properties of the mechanical arm, the above limitations may lead to inaccurate solutions during the solving process.

2.2 Algebraic Method

The algebraic method is to directly combine the items in the pose matrix T0, T1, …, Tn, and to define the intermediate variables (which is a combination of joint parameters). For the sake of constructing a one-dimensional high-order equation. All joint angles can be found by solving this equation. We use the same example as shown in Figure 1, the pose matrix of the end point is.

$${}^{0}_{3}T = \begin{bmatrix} \cos(\theta_1+\theta_2+\theta_3) & -\sin(\theta_1+\theta_2+\theta_3) & 0 & l_1\cos\theta_1 + l_2\cos(\theta_1+\theta_2) \\ \sin(\theta_1+\theta_2+\theta_3) & \cos(\theta_1+\theta_2+\theta_3) & 0 & l_1\cos\theta_1 + l_2\cos(\theta_1+\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$= \begin{bmatrix} \cos(\emptyset) & -\sin(\emptyset) & 0 & x \\ \sin(\emptyset) & \cos(\emptyset) & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \ . \tag{4}$$

By (3) and (4), all angles could be solved by simple algebraic computations.

We refer to some research findings [9-12], which show that algebraic method can be used in combination with geometric method to control the motion and posture of the robot arm. By using the kinematic model and inverse kinematic algorithm, algebraic method can calculate the joint angles of the robot arm to achieve the desired motion path. It is used to solve the position and posture of the robot arm in space. For example, in the study by Serdar Kiiqiik et al. [16], we can learn that the algebraic method usually involves building the kinematic model of the robot arm, which requires knowledge of the function relationship between the joint angles and gear ratios. The target position and posture are then input into the kinematic model to calculate the equation for inverse kinematics, which can be solved to control the motion of the robot arm by determining the joint angles. The algebraic method directly combines the pose matrices T0, T1, Tn and defines intermediate variables (which are combinations of joint parameters) to construct a one-dimensional higher-order equation. All joint angles can be found by solving this equation. We use the same example as shown in Figure 1, with the end pose matrix as follows: This method can be used to solve various aspects of inverse kinematics problems for the robot arm, including kinematic model, inverse kinematic equations, analytical and numerical solutions, volume constraints, and collision avoidance detection, etc. We can use these methods to design and control high-precision and high-efficiency robot arms. However, since the problem may not have a unique solution or a solution at all, specific algorithms or strategies are required to solve it. Numerical solutions may have convergence problems and may require a lot of time and resources to calculate the best strategy for each posture. As

the kinematic model becomes more complex, solving the inverse kinematic equations becomes more difficult.

2.3 Neural Networks

Due to the complexity of inverse kinematics and the need to consider multiple sources of uncertainty, neural networks (NNs) have been used to solve inverse kinematics problems. NNs can handle high-dimensional input data and nonlinear relationships, and in many applications, NNs can find the optimal solution faster than other methods. Ahmed R. J. Almusawi et al. proposed a new solution for the inverse kinematics of robotic arms based on an artificial neural network (ANN) architecture [17]. The motion of the robot arm is controlled by the ANN based on its kinematics. The novelty of this ANN-based inverse kinematics method is that the input pattern to the neural network includes feedback on the current joint angle configuration of the robot arm and the desired position and direction. Adding the current configuration of joint angles to the ANN input greatly improves the accuracy of the estimated output of joint angles. For example, Nurettin Gökhan ADAR proposed a method that combines a proportional-integral (PI) control algorithm with a neural network model [18]. This method involves developing a multilayer feedforward neural network model to solve the inverse kinematics problem of a 5-degree-offreedom robotic arm, and uses PI control in combination with the ANN model as the algorithm. This approach has stronger tracking ability, smaller control errors, and better absolute fit with reference values, allowing for real-time position control of the robot arm. Many studies have found that [19-24], when compared with existing analytical techniques, using NNs has advantages in minimizing position error and estimating joint angle accuracy. For inverse kinematics problems, by training a neural network to learn the inverse kinematics relationship, inputting feedback on the current joint angle configuration and the desired position and direction, the inverse kinematics problem can be quickly solved, allowing for adaptive control, planning, and improved performance of the robot arm.

2.4 Reinforcement Learning

Reinforcement learning is a method that improves strategies through interaction with the environment. Starting from an initial random strategy, it continually experiments, adjusting the strategy based on the rewards or penalties received, with the aim of finding a policy that maximizes the reward. This is the approach proposed by Peters and Schaal for using reinforcement learning in robot arm operation control [25]. They define the strategy as the actions of the robot arm when given a target position or posture, and the reward is defined as the proximity to which the robot arm achieves the target. This allows the robot arm to learn how to effectively reach a specific position or posture.

However, the process of reinforcement learning can require a substantial amount of time and resources. This process requires a large amount of trial and error to find an effective strategy and does not always guarantee that the optimal strategy will be learned, thus convergence might not be superior. This problem is also mentioned in the research by Weber and Schmidt [26]. In addition, designing an appropriate reward function is of utmost importance for reinforcement learning. If the reward function is improperly designed, the strategy learned by the robot arm might significantly deviate from the expected strategy.

2.5 Summary

Controlling a robotic arm requires solving the forward and inverse kinematics problems to coordinate positions in Cartesian space. In real-time control applications, inverse kinematics can be relatively difficult due to high computational requirements and long running times. Traditional solving methods, such as using geometric and algebraic techniques, are insufficient and slow in the process of inverse kinematics problem-solving.

3 Particle Filter and Application in Inverse Kinematics

When using algebraic and geometric methods, algebraic methods require solving complex equation combinations, while geometric methods may be limited by analytical solutions. Using ANN to solve inverse kinematics problems is an effective method, but due to the high complexity of robot kinematics, the ANN model requires a large amount of computing resources and sufficient training data, which undoubtedly increases time and cost. In recent years, using particle swarm optimization to solve the problem of robot arm path planning has attracted much attention, but due to the different algorithmic designs, the particle swarm optimization has its own advantages and disadvantages in solving different problems. Therefore, this study proposes a more balanced and less costly solution. The particle filter we designed not only calculates the distance of the robot arm end target, but also calculates the change in arm posture. In continuous movement problems, the particle filter selects the particle end coordinates with the smallest current posture change and calculates the next end coordinates, thus solving the problem of oscillation caused by rapid posture changes. Geometric and algebraic methods can usually calculate all possible solutions at the current moment, but they cannot determine the optimal solution. However, when using the particle filter to solve the current posture problem, the best solution can be selected by choosing the smallest posture change between ADJACENT solution points.

3.1 Particle Filter

Particle filters [27-30] are commonly used in robot localization and tracking applications. In this study, we use particles in a particle filter to represent the pose of the robotic arm, with the position of each particle indicating the end effector position of the arm. Each particle can be considered as a state of the robotic arm, with the joint angles embedded in the particle parameters. A set of particles is used to track the position of the robotic arm's end effector, and the poses of the particles are averaged to find the estimated pose. Through this approach, the inverse kinematics of the robotic arm can be effectively computed. In a particle filter, a particle is a tuple of n variables $p = (v_1, v_2, \ldots, v_n)$, where v_i is a state of the robot, such as the joint angles and end effector position of a robotic arm, or the position coordinates, velocity,

acceleration, and orientation of an autonomous vehicle. In short, a particle set is a collection of hundreds, thousands, or even tens of thousands of particles, which is what we call a particle filter summary. A particle filter is a type of Bayesian filter, specifically, a nonparametric Bayesian filter. The Bayesian filter is an algorithm that estimates the state of a robot based on the Bayes theorem. In this study, we refer to the robot as a multi-joint robotic arm. The end effector position of the robotic arm can be indirectly computed from the joint angles and link lengths.

3.2 Design of Particle Filter in Inverse Kinematics

The main purpose of this section is to use the particle filter to deal with the inverse kinematics of the robot arm. The so-called inverse kinematics is to find an attitude under the condition that the end coordinates of the robot arm are known, so that the end coordinates of the robot arm under the attitude can meet the condition. The attitude of the robot arm is determined by all the joint angles together. Therefore, in more detail, the so-called inverse kinematics is to derive the angle of each joint from the known terminal coordinates, so that the terminal coordinates of the robot arm can meet the required conditions under the attitude.

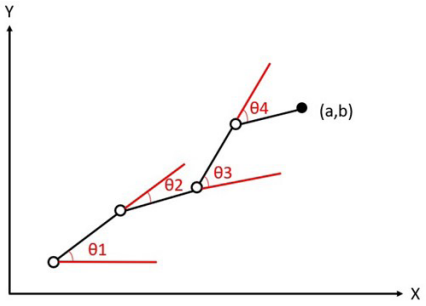

Figure 2. Four joints robot arm

In Figure 2, the angle of each joint is calculated from the extension of the previous link to the angle of the link of the joint. The first joint is calculated from the horizontal line to the angle of the first link. As we described in the previous section, particle filters are generally used in robot positioning and tracking problems. We can consider the process of finding the appropriate attitude of the robot arm as a localization problem. On the other hand, we can also consider this problem as a tracking problem. The details of the particle filter algorithm are explained as below.

3.2.1 Step 1: Random Particle Generation

The first step is to randomly generate particles. The following symbols are used to represent particles.

$$\begin{matrix}(\theta_1^{(1)},\theta_2^{(1)},\theta_3^{(1)},\theta_4^{(1)})\\(\theta_1^{(2)},\theta_2^{(2)},\theta_3^{(2)},\theta_4^{(2)})\\\vdots\\(\theta_1^{(n)},\theta_2^{(n)},\theta_3^{(n)},\theta_4^{(n)})\end{matrix} \quad (5)$$

The lower number represents the number of joint angles, while the upper number represents the number of particles.

3.2.2 Step 2: Calculate the Importance Weight

After generating random particles, the next step is to calculate the importance weight for each particle. The weight of a particle represents how good it is. The Figure 3 shows the components of the error function.

```
0  for i in range(self.num_particles):
1      errors[i][0] = (locations[i][0]-goal[0])**2 + (locatios[i][1]-goal[1])**2
2      errors[i][0] = errors[i][0] + (self.particles[i][0]-joints[0]**2
3      errors[i][0] = errors[i][0] + (self.particles[i][1]-joints[1]**2
4      errors[i][0] = errors[i][0] + (self.particles[i][2]-joints[2]**2
5      errors[i][0] = errors[i][0] + (self.particles[i][3]-joints[3]**2
```

Figure 3. Composition of the ERRORS function

The function consists of two parts: the first part calculates the distance between the target and the particle, and the second part calculates the distance between the mechanical arm's each link (Link 1 to Link 4) and the target. To calculate the distance error between the target and the particle, we need the joint components $(x^{(i)}, y^{(i)})$ of the particle. Then we calculate the distance between the calculated end coordinates and the actual end coordinates (x_0, y_0). Each particle represents a certain posture, and an end coordinate can be obtained using forward kinematics. Then,

$$(x^{(i)}, y^{(i)}) = FK(\theta_1^{(i)}, \theta_2^{(i)}, \theta_3^{(i)}, \theta_4^{(i)}) \ . \quad (6)$$

The usual Euclidean distance, $d^{(i)}$, represents the distance from the i^{th} particle to (x_0, y_0). The distance,

$$d^{(i)} = \sqrt{(x^{(i)} - x_0)^2 + (y^{(i)} - y_0)^2} \ . \quad (7)$$

Once we have the distance, we can convert it to a weight:

$$w^{(i)} = \frac{1}{1 + d^{(i)}} \ . \quad (8)$$

3.2.3 Step 3: Resampling

The weight of each particle calculated in the previous step will be used as the basis for resampling. We convert the weights into probabilities, and then use the probabilities to resample the particles. Therefore, the higher the weight of a particle, the more often it will be selected after resampling. Define

$$W = \sum_{i=1}^{n} w^{(n)} \ . \quad (9)$$

Then,

$$\overline{w}^{(i)} = \frac{\overline{w}^{(i)}}{W} = p^{(i)} \ . \quad (10)$$

The probability here will be the probability that the particle will be selected after resampling.

3.2.4 Step 4: Solve the New Pose and Return to Step 2

Finally, all the resampled particles are applied to the forward kinematics to obtain a group of end coordinates. This group of end coordinates is averaged to get an estimation of the current end coordinates. After the calculation is completed, return to Step 2 for the next estimation.

4 Experimental Results

A 4-link robotic arm in 2D coordinates is implemented via Python language that is shown in Section 4.1. It also facilitates the adjustment and verification of parameters in the simulation program, such as the sigma value and the calculation parameters of errors. In Sections 4.2, we let the robotic arm draw some equations to confirm its execution effect. The experiment starts from simple graphics and gradually progresses to more complex ones.

4.1 Simulator Design

In order to test the effectiveness of the method proposed in this paper, we designed a software robotic arm simulator. This design is based on an object-oriented design approach. Any planar robotic arm structure can be generated by the RobotArm class. The RobotArm category is composed of the Link class. We use the bottom-up approach to design the Link class first, and its attributes are:

- length: the length of the link
- joint_x: The joint of this bar (link) Coordinate
- joint_y: the joint of this bar (link) Coordinate
- angle: The angle of the joint
- end_x: Terminal coordinates
- end_y: terminal coordinates

This class has a member function moveto() to move the bar. The class RobotArm can be defined using the above Link class. If we take a four-rod planar robot arm, the Python language defines it as follows.

```
class RobotArm:
    def __init__(self, x, y, theta1, theta2, theta3, theta4, L1, L2, L3,
                 L4):
        self.link1 = Link(x, y, theta1, L1, 'black',20)
        self.link2 = Link(self.link1.end_x, self.link1.end_y,
                          theta1+theta2, L2, 'blue',15)
        self.link3 = Link(self.link2.end_x, self.link2.end_y,
                          theta1+theta2+theta3, L3, 'green',10)
        self.link4 = Link(self.link3.end_x, self.link3.end_y,
                          theta1+theta2+theta3+theta4, L4, 'red',5)
    def moveto(self, theta1, theta2, theta3, theta4, penState=True):
        self.link1.moveto(self.link1.joint_x, self.link1.joint_y,
                          theta1, False)
        self.link2.moveto(self.link1.end_x, self.link1.end_y,
                          theta1+theta2, False)
        self.link3.moveto(self.link2.end_x, self.link2.end_y,
                          theta1+theta2+theta3, False)
        self.link4.moveto(self.link3.end_x, self.link3.end_y,
                          theta1+theta2+theta3+theta4, penState)
```

In the RobotArm class function, some parameters need to be passed when initialized. Among them, x and y represent the starting position on the coordinate system, theta1 to theta4 represent the angles of link1 to link4, and L1 to L4 represent the lengths of link1 to link4. The self.link1 represents the end position of link1, which starts at the coordinate (x, y) and moves in the direction of theta1 with the length of L1. Similarly, self.link2 represents the end position of link2, which starts at the end position of link1 and moves in the direction of theta1+theta2 with the length of L2. The same goes for self.link4, which represents the end position of link4, the end effector of the robot arm.

In the member function moveto, some parameters need to be passed when used. theta1 to theta4 represent the angles of link1 to link4. self.link1.moveto moves link1 to its end position starting from its initial position, and adds the angle theta1 to get the end position. Similarly, self.link2.moveto moves link2 to its end position starting from the end position of link1, and adds the angle theta1+theta2 to get the end position. The same goes for self.link4, which moves the end effector of the robot arm to its end position. The parameter penState behind it indicates whether to display it, which means moving the robot arm to the end position. Next, we explain how to use this simulated arm. As shown in Figure 4, we create the arm and set its starting position. Assuming the initial coordinates are (-200, -200), the figure has 4 links (Link1 to Link4), each with a length of 100 and an angle of 0 degrees, so its end coordinates are (200, -200).

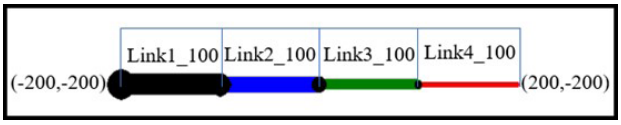

Figure 4. Setting the initial position of the robotic arm simulator at (-200, -200), with the end coordinates being (200, -200)

4.2 Experimental Design

Using the four links of the robotic arm (Link1 to Link4), we conducted experiments to make the arm draw equations. First, the program randomly generates particles, which represent the positions in different postures that the robotic arm can reach, as shown in Figure 5. Next, the program calculates the error for each particle, which is the difference between the position and posture of the robotic arm at the starting point and the particle. The error is then transformed into weight. The robotic arm then moves to the particle with the highest weight. The movement of the robotic arm consists of two stages, as described below for each equation design:

4.2.1 Experiment 1

Draw a straight-line equation y=-x-50 (as shown in Figure 6). In the first stage of the robotic arm movement, it moves from the starting position to the position for drawing, as shown in Figure 7. In the second stage, it starts drawing from the drawing position, as shown in Figure 8.

4.2.2 Experiment 2

Draw a quadratic curve $y= x^2-2$ (as shown in Figure 9). In the first stage of the robotic arm movement, it moves from the starting position to the position for drawing, as shown in Figure 10. In the second stage, it starts drawing from the drawing position, as shown in Figure 11.

4.2.3 Experiment 3

Draw a cubic curve equation $y = x^3-2x^2$ (as shown in Figure 12). In the first stage of the robotic arm movement, it moves from the starting position to the position for drawing, as shown in Figure 13. In the second stage, it starts drawing from the drawing position, as shown in Figure 14.

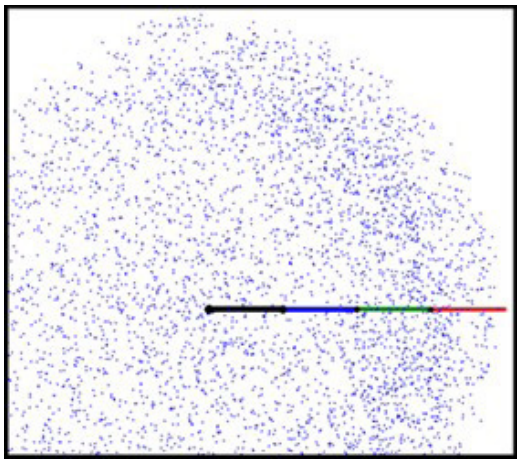

Figure 5. Randomly generate particles, which represent different poses that the robot arm can reach

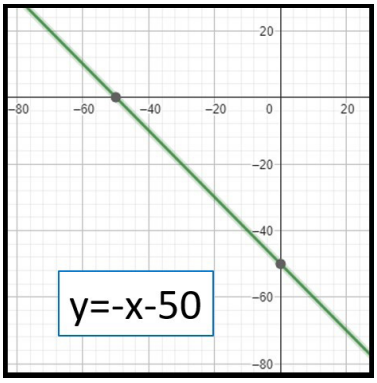

Figure 6. Let the robot arm draw the graph with the equation y = -x-50

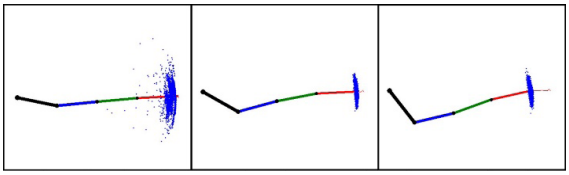

Figure 7. After calculating the weight of each particle, move the robot arm to the particle with the highest weight

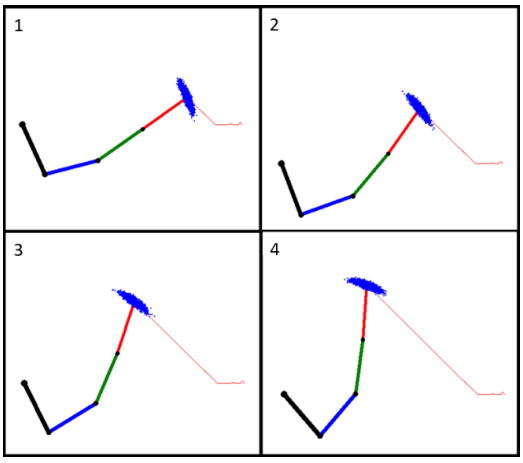

Figure 8. Move the robot arm to the particle with the highest weight (Draw a straight line)

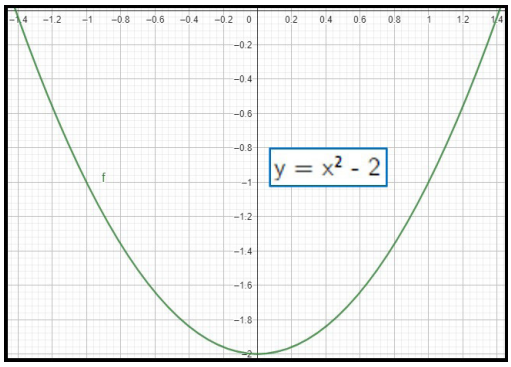

Figure 9. Have the robotic arm draw the graph of the equation $y = x^2-2$

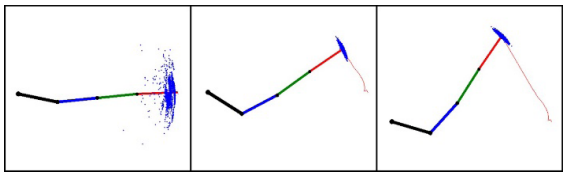

Figure 10. After calculating the weight of each particle, move the robot arm to the particle with the highest weight

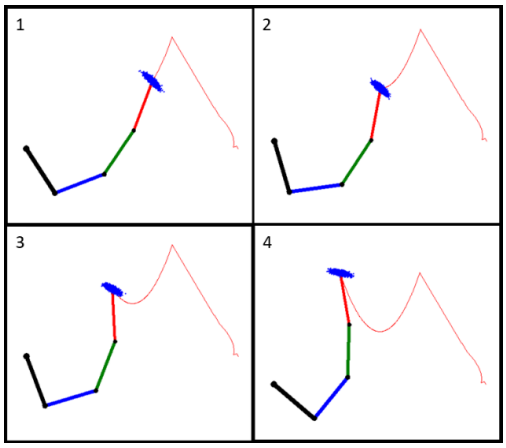

Figure 11. Move the robotic arm to the particle with the highest weight (Draw a quadratic curve $y = x^2-2$)

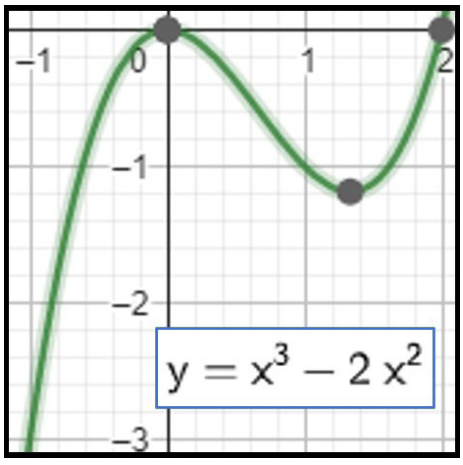

Figure 12. Have the robotic arm draw the graph of the equation $y = x^3-2x^2$

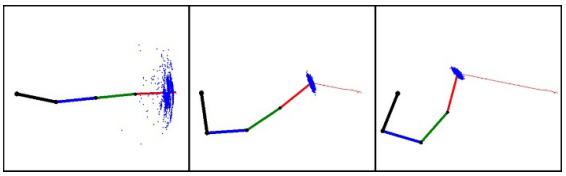

Figure 13. After calculating the weight of each particle, move the robot arm to the particle with the highest weight

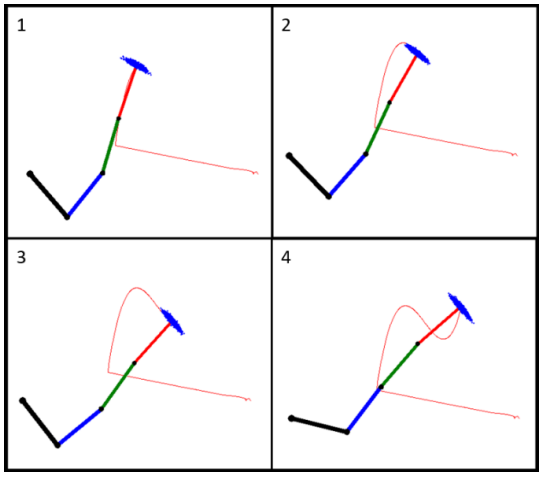

Figure 14. The robot arm moves to the position with a high weight particle (Draw a cubic curve equation $y = x^3 - 2x^2$)

4.2.4 Summary

The results of this experiment show that there is a convergence issue at the beginning when using the particle filter, but it gradually converges. Once it converges and moves to any other reasonable position, there will be no more oscillation. By using the particle filter to calculate the distance between particles and the target, as well as the posture of each link of the robot arm, the oscillation problem caused by multiple solutions can be solved. From the experimental results, we can see that when the robot arm adopts a combination of particle filtering and algorithms with a large number of random sampling points and importance weights, the operation of the arm is smooth without any oscillation.

Finally, a comparison was given on some commonly used methods for inverse kinematics, as shown in Table 1.

Table 1. A comparison on some commonly used methods for IK

Method	Time complexity	Space complexity	Convergence	ClosedForm derivation	Multiple solutions issue
Ge	low	low	no issue	difficult	can't
Al	low	low	no issue	difficult	can't
NN	high	high	difficult	easy	can't
RL	high	medium	difficult	medium	can't
PF	medium	medium	easy	easy	can

Ge (Geometric)

Fast computational speed, require less memory resources, and do not have convergence issues. However, the derivation of Closed-Form is difficult.

Al (Algebraic)

Just like the Ge method, the derivation of Closed-Form is difficult.

NN (Neural Networks)

The computation is more complex and slower, requiring more memory resources. convergence is difficult.

RL (Reinforcement Learning)

Similar to the NN method, convergence is difficult.

PF (Particle Filter)

Computation is not complex, and the speed is acceptable. It doesn't consume much memory resources. There is no need to deduce the Closed-Form, and there will be no divergence problem.

5 Conclusions

A common problem encountered in the inverse kinematics of mechanical arms is the presence of multiple solutions. These multiple solutions often appear symmetrically. To avoid the interference of symmetric multiple solutions, we have abandoned the traditional algebraic or geometric inverse kinematics approach and adopted the Monte Carlo-based non-parametric Bayesian filter (also known as the particle filter) to address the inverse kinematics problem of the robotic arm. The particle filter is designed with a large number of random sample points and importance weights, allowing the sample points to converge to the optimal solution through an iterative process. This paper illustrates how the design of importance weights can be used to avoid the oscillation problem of symmetric multiple solutions in adjacent time samples during robot motion. Future research will inevitably require goals for robotic arms operating in three dimensions, and we will be working towards addressing the challenges associated with three-dimensional robotic arms.

References

[1] M. P. Mills, How Robots Will Transform the 2020s, *Reason*, Vol. 53, No. 11, pp. 37-41, April, 2022.

[2] K. C. Lan, G. Litscher, Robot-controlled acupuncture-an innovative step towards modernization of the ancient traditional medical treatment method, *Medicines*, Vol. 6, No. 3, Article No. 87, September, 2019.

[3] W. Montalvo, J. Escobar-Naranjo, C. A. Garcia, M. V. Garcia, Low-cost automation for gravity compensation of robotic arm, *Applied Sciences*, Vol. 10, No. 11, Article No. 3823, June, 2020.

[4] K. Tai, A. R. El-Sayed, M. Shahriari, M. Biglarbegian, S. Mahmud, State of the art robotic grippers and applications, *Robotics*, Vol. 5, No. 2, Article No. 11, June, 2016.

[5] Y. Wei, S. Jian, S. He, Z. Wang, General approach for inverse kinematics of nR robots, *Mechanism and Machine Theory*, Vol. 75, pp. 97-106, May, 2014.

[6] M. Carricato, Direct geometrico-static problem of underconstrained cable-driven parallel robots with three cables, *Journal of Mechanisms and Robotics*, Vol. 5, No. 3, Article No. 031008, August, 2013.

[7] K. Raza, T. A. Khan, N. Abbas, Kinematic analysis and geometrical improvement of an industrial robotic arm, *Journal of King Saud University-Engineering Sciences*, Vol. 30, No. 3, pp. 218-223, July, 2018.

[8] A. Goldenberg, B. Benhabib, R. Fenton, A complete generalized solution to the inverse kinematics of robots, *IEEE Journal on Robotics and Automation*, Vol. 1, No. 1, pp. 14-20, March, 1985.

[9] S. Kucuk, Z. Bingul, Inverse kinematics solutions for industrial robot manipulators with offset wrists, *Applied Mathematical Modelling*, Vol. 38, No. 7-8, pp. 1983-1999, April, 2014.

[10] I. M. Chen, Y. Gao, Closed-form inverse kinematics solver for reconfigurable robots, *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, Seoul, South Korea, 2001, Vol. 3, pp. 2395-2400.

[11] A. Morell, M. Tarokh, L. Acosta, Inverse kinematics solutions for serial robots using support vector regression, *2013 IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, 2013, pp. 4203-4208.

[12] W. Guo, R. Li, C. Q. Cao, Y. F. Gao, Kinematics analysis of a novel 5-DOF hybrid manipulator, *International Journal of Automation Technology*, Vol. 9, No. 6, pp. 765-774, September, 2015.

[13] M. Bi, Control of robot arm motion using trapezoid fuzzy two-degree-of-freedom PID algorithm, *Symmetry*, Vol. 12, No. 4, Article No. 665, April, 2020.

[14] Z. Mu, H. Yuan, W. Xu, T. Liu, B. Liang, A segmented geometry method for kinematics and configuration planning of spatial hyper-redundant manipulators, *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Vol. 50, No. 5, pp. 1746-1756, May, 2020.

[15] S. Yahya, M. Moghavvemi, H. F. Mohamed, Geometrical approach of planar hyper-redundant manipulators: Inverse kinematics, path planning and workspace, *Simulation Modelling Practice and Theory*, Vol. 19, No. 1, pp. 406-422, January, 2011.

[16] S. KuCuk, Z. Bingul, The inverse kinematics solutions of industrial robot manipulators, *Proceedings of the IEEE International Conference on Mechatronics*, *2004. ICM'04*, Istanbul, Turkey, 2004, pp. 274-279.

[17] A. J. Almusawi, L. C. Dülger, S. Kapucu, A new artificial neural network approach in solving inverse kinematics of robotic arm (denso vp6242), *Computational intelligence and neuroscience*, Vol. 2016, No. 1, pp. 1-10, January, 2016.

[18] N. G. Adar, Real Time Control Application of the Robotic Arm Using Neural Network Based Inverse Kinematics Solution, *Sakarya University Journal of Science*, Vol. 25, No. 3, pp. 849-857, June, 2021.

[19] A. V. Duka, Neural network based inverse kinematics solution for trajectory tracking of a robotic arm, *Procedia Technology*, Vol. 12, pp. 20-27, 2014.

[20] R. Köker, C. Öz, T. Çakar, H. Ekiz, A study of neural network based inverse kinematics solution for a three-joint robot, *Robotics and autonomous systems*, Vol. 49, No. 3-4, pp. 227-234, December, 2004.

[21] J. Lu, T. Zou, X. Jiang, A Neural Network Based Approach to Inverse Kinematics Problem for General Six-Axis Robots, *Sensors*, Vol. 22, No. 22, Article No. 8909, November, 2022.

[22] S. G. Shuzhi, C. C. Hang, L. C. Woon, Adaptive neural network control of robot manipulators in task space, *IEEE transactions on industrial electronics*, Vol. 44, No. 6, pp. 746-752, December, 1997.

[23] Y. Peng, Z. Peng, T. Lan, Neural Network Based Inverse Kinematics Solution for 6-R Robot Implement Using R Package Neuralnet, *2021 5th International Conference on Robotics and Automation Sciences (ICRAS)*, Wuhan, China, 2021, pp. 65-69.

[24] J. Junior, R. Jesus, L. Molina, E. Carvalho, E. O. Freire, FRPSO: Inverse kinematics using fully resampled particle swarm optimization, *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, João Pessoa, Brazil, 2018, pp. 402-407.

[25] J. Peters, S. Schaal, Reinforcement learning by reward-weighted regression for operational space control, *Proceedings of the 24th international conference on Machine learning*, Corvalis, Oregon USA, 2007, pp. 745-750.

[26] J. Weber, M. Schmidt, An improved approach for inverse kinematics and motion planning of an industrial robot manipulator with reinforcement learning, *2021 Fifth IEEE International Conference on Robotic Computing (IRC)*, Taichung, Taiwan, 2021, pp. 10-17.

[27] M. S. Arulampalam, S. Maskell, N. Gordon, T. Clapp, A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking, *IEEE Transactions on signal processing*, Vol. 50, No. 2, pp. 174-188, February, 2002.

[28] M. Jouin, R. Gouriveau, D. Hissel, M. Pera, N. Zerhouni, Particle filter-based prognostics: review, discussion and perspectives, *Mechanical systems and signal processing*, Vol. 72-73, pp. 2-31, May, 2016.

[29] Z. Jiang, W. Zhou, H. Li, Y. Mo, W. Ni, Q. Huang, A new kind of accurate calibration method for robotic kinematic parameters based on the extended Kalman and particle filter algorithm, *IEEE Transactions on Industrial Electronics*, Vol. 65, No. 4, pp. 3337-3345, April, 2018.

[30] C. L. Chiang, I. L. Lin, C. C. Hsieh, Y. Y. Chiang, M. H. Yang, Using Particle Filter to Solve Problem of Symmetric Multiple Solutions in Inverse Kinematics of Manipulator, *2022 IEEE 4th Eurasia Conference on IOT, Communication and Engineering (ECICE)*, Yunlin, Taiwan, 2022, pp. 599-601.

Biographies

Chien-Lin Chiang is currently a doctoral student at Tatung University. Since 2022, he has served as a lecturer at Vanung University. He is also the secretary of the Taiwan Association for Digital Forensics Development (ACFD) and the Vice General Manager of Baofeng Plastics Company. His research areas include artificial intelligence, computer hardware integration, blockchain applications, and information security. He is dedicated to promoting automation and transformation in traditional industries, as well as industrial safety applications in his field.

Chang-Chen Hsieh received the B.S. degree from Yuan Ze University, Taiwan, in 2002, and the M.E. degree from Vanung University, Taiwan, in 2023, Since 2023, he has served as a lecturer at Vanung University, Taiwan. His research interests are machine learning and robotic system integration.

Yi-Yuan Chiang is an Assistant Professor at Vanung University, Taiwan, where he has been a part of the faculty since 2008. He received his Ph.D. in 2007 from Yuan Ze University, Taiwan. His research focuses on machine learning, sensor fusion, and robot systems integration. Notably, he has made significant contributions to the field and holds numerous patents for his innovative work.

I-Long Lin received his BS degree in public security and law from National Central Police University, Taoyuan, Taiwan in 1983, and the MS degree in computer science and information engineering from Tamkang University, Tamsui, Taiwan in 1989. He received the PhD degree in computer and information science from National Taiwan University of Science and Technology in July 1997. He was a Professor of Yuan Pei University of Medical Technology from 2011 to 2020, and He is currently a Professor of Tatung University and director of Research Center on Cybercrimes, Digital Evidence and Forensic Computing of the University, and director of Research Center on Cloud Service and Cyber Forensics of the University.