IKDP :
Inverse Kinematics through Diffusion Process

Hao-Tang Tsui*
College of ECE
National Yang Ming Chiao Tung University
henrytsui000@gmail.com

Yu-Rou Tuan*
College of ECE
National Yang Ming Chiao Tung University
yztuan1129@gmail.com

Hong-Han Shuai
College of ECE
National Yang Ming Chiao Tung University
hhshuai@nycu.edu.tw

Abstract—It is a common problem in robotics to specify the position of each joint of the robot so that the end point reaches a certain target in space. This can be solved in two ways:
1. Forward kinematics method (FK)
2. Inverse kinematics method (IK)

However, inverse kinematics cannot be solved by an algorithm. The common method is the Jacobian inverse technique, and some people have tried to find the answer by machine learning. And in this project, we will show how to use the Conditional Denoising Diffusion Probabilistic Model (DDPM) [1] to integrate the solution of calculating IK.

Index Terms—Inverse kinematics, Denoising Diffusion Probabilistic Model, self-Attention, Transformer

I. INTRODUCTION

The motivation for doing this topic comes from our side project. We found that the inverse kinematics method(IK), which is commonly used to control the movement of human models in game engines, has a lot to be improved, such as the fidelity of movement and the angle of joint rotation accuracy. Therefore, we decided to do research on it.

Inverse kinematics is used to determine the kinematic equations to motion of a robot to reach a desired position. For example, given a 3D model of a human body, if the goal is to change the hand from a relaxed position to a waving position, we can use inverse kinematics to find the angle of the wrist and elbow.

Therefore, we hope to convert the IK problem into an objective function in machine learning, combined with the denoising diffusion probabilistic models(DDPM) [1] taught this semester, consider the target condition that will be needed in the IK process, so our topic will use conditional DDPM to find IK's solution.

That is to train a model F with given target position (t_x, t_y), so that $F(t_x, t_y)$ has the best solution θ, θ is a series of robot joint angles, so that the end of the robot can reach (t_x, t_y). Because we expect the robot's hand to reach a specific target, the target (t_x, t_y) is added to the time embedding and becomes a condition diffusion form.

II. RELATED WORK

A. Inverse Kinematics Method (IK)

Inverse kinematics method (IK) is responsible for calculating the joint movement required for the end position. It will
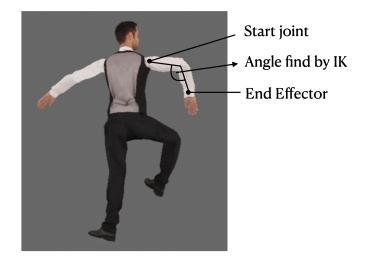

Fig. 1: Pose Estimation Calculation for the depth of joints

calculate the angle of each joint from a terminal joint and the length of each bone to form a robotic arm that touches this distal point. At present, IK can be realized through quite a variety of algorithms, such as Jacobian method [3] and machine learning methods that have been gradually proposed in recent years [2]. To sum up, we think that IK is worth improving, so we plan to use machine learning to improve it.

B. Denoising Diffusion Probabilistic Models (DDPM)

Denoising Diffusion Probabilistic Models (DDPM), as mentioned in the class, continuously removes the noise from a gaussian noise through the encoder and decoder until it becomes the image we want to generate. DDPM can also be done in 1D, which is our topic's task. We would like to use this technology because the task of IK is also related to generation. We want to achieve the task of generating the angle of the joint point of the manipulator through Conditional DDPM [4].

III. METHODS

A. Ideas

Design a diffusion model $F = (t_x, t_y, t_z)$, given the target position (t_x, t_y, t_z) of the distal joint t and the direction of the solution bone vector (two adjacent joints form a bone vector).

With known bone length, we define that there are total N joints on a robotic arm, so there will be N bone vectors. Finally, the bone vectors will be combined into a vector list $\vec{A} = \{\vec{A^0}, \vec{A^1}, ..., \vec{A^{N-1}}\}$ of the best solution, and the end of the robotic arm can reach target position (t_x, t_y, t_z), shown as Fig. 2. In three-dimensional space, each bone vector $\vec{A^n}$ has three components x, y, and z. We define that $\vec{A_x^n}$ is the x component of the n_{th} bone, y and z components can be deduced by analogy. See equation (1).

$$t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \sum_{n=0}^{N-1} \vec{A^n} = \sum_{n=0}^{N-1} \begin{bmatrix} \vec{A_x^n} \\ \vec{A_y^n} \\ \vec{A_z^n} \end{bmatrix} \quad (1)$$
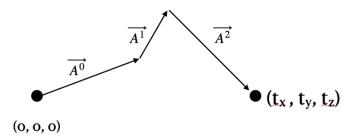

Fig. 2: Robotic arm vector schematic diagram

We found that instead of calculating these bone vectors from position directly, computing the angular ratio vectors of the joints simplifies the problem. Which means to calculate the angle between the bone vectors and the standard coordinate system, because the length of these bone vectors is fixed. See equation (2) and Fig. 3, θ^n represents the angle between the n_{th} joint and the $+x-$axis, and b^n represents the length of the n_{th} bone vector. The sum of the cumulative rotations of the x, y, and z axes of each joint will be the position (t_x, t_y, t_z) of the robotic arm's target t.

$$t = \sum_{n=0}^{N-1} \begin{bmatrix} b^n \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} \cos\theta_y^n & 0 & \sin\theta_y^n \\ 0 & 1 & 0 \\ -\sin\theta_x^n & 0 & \cos\theta_x^n \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x^n & -\sin\theta_x^n \\ 0 & \sin\theta_x^n & \cos\theta_x^n \end{bmatrix} \begin{bmatrix} \cos\theta_z^n & -\sin\theta_z^n & 0 \\ \sin\theta_z^n & \cos\theta_z^n & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$

Then we simplify the problem to only two dimensions. The position of target t is (t_x, t_y), and each bone length has a length of 1, shown as equation (3). Our task becomes to design a model F so that $F(t_x, t_y)$ obtains best solution $\hat{\theta} = \{\hat{\theta^0}, \hat{\theta^1}, ..., \theta^{\hat{N-1}}\}$. See below equation (4), after substituting the bone vector of $(1, 0)$, it shows that t_x in the target position is the sum of cosine values of each joint points' rotation angles, and t_y is the sum of their sine value. $(hatt_x, hatt_y)$, that is the end joint that obtained by the model, can be calculated by $\hat{\theta^n}$ in the same way. If we want to go back to the original question of 3-dimension, we only need to add another dimension from the sum to the rotated length of the joint angle and the inner product of the bone length.

$$t = \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \sum_{n=0}^{N} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} \cos\theta^n & \sin\theta^n \\ -\sin\theta^n & \cos\theta^n \end{bmatrix} \quad (3)$$

$$t_x = \sum_{n=0}^{N} \cos\theta^n, \quad t_y = \sum_{n=0}^{N} \sin\theta^n \quad (4)$$

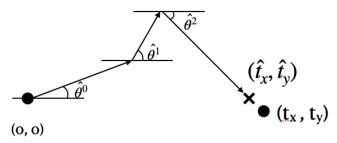$$\hat{t_x} = \sum_{n=0}^{N} \cos\hat{\theta^n}, \quad \hat{t_y} = \sum_{n=0}^{N} \sin\hat{\theta^n} \quad (5)$$

Fig. 3: Pose Estimation Calculation for the depth of joints

B. Propose

Train a Conditional DDPM model to gradually denoise from θ_T to $\hat{\theta}_0$. Where $\theta_T = \{\theta | N(\mu, \sigma)\}$ represents a list of noise sample from Gaussian, and $\hat{\theta}_0 = \{\hat{\theta_0^0}, \hat{\theta_0^1}, ... \theta_0^{\hat{N-1}}\}$ represent the angle to be rotated by each joint. $\hat{\theta}_0$ should be close to the ground truth θ_0, make $\hat{t} = (\hat{t_x}, \hat{t_y})$ close to ground truth $t = (t_x, t_y)$.

In Inverse Kinematics task, it is not enough to use only the diffusion model, because we also need to turn the tip to the target, so we will add condition t_x, t_y to indicate where the tip of the robot arm should point. That is, we hope that $\hat{t}$ is close to t.
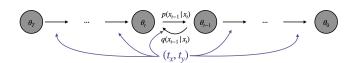

Fig. 4: Process of training conditional DDPM model

C. Loss Function

Evaluating the performance of Inverse Kinematics method will need to calculate the distance or difference, so we think Mean-Square Error(MSE) will be a good choice. In our model, there are a total of two losses as indicators for evaluate the performance of the model, and we expect both losses are as low as possible. The two matrices we evaluated, namely:

1) Distance of t: $||t - \hat{t}||_2 = \sqrt{(t_x - \hat{t_y})^2 + (t_y - \hat{t_y})^2}$ Evaluates the quality of the model, and represents the differences of the end joint of the robotic arm and target position.

2) Loss of θ: $||\theta - \hat{\theta}||_2 = \sqrt{\Sigma_{n=0}^{N-1}(\theta^n - \hat{\theta^n})^2}$ Represents the difference between each joint point and ground truth, and it is the loss function which would do backward.

D. Model Architecture

In a timestep, we hope that the input to the model is $\theta_{t-1} = \{\theta^0_{t-1}, \theta^1_{t-1}, ..., \theta^{N-1}_{t-1}\}$, which is a list indicated how many degrees each joint of the robotic arm should turn. After input through a transformer encoder, we can obtain the feature. Concate the feature with the time embedding and condition which is the target position (t_x, t_y). Its output will pass through a few layers of MLP, finally, go through the transformer decoder, and add the residual of the previous θ_{t-1} to become the θ_t we want, that is, the angle of each joint after 1 timestep of denoising process. The model architecture diagram is Fig. 5
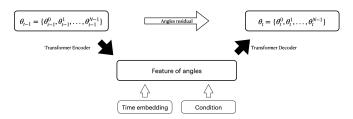

Fig. 5: Pose Estimation Calculation for the depth of joints

E. Training Process

Algorithm 1: Pseudo-code of training IKDP.

Input: A list of ground truth θ
1 Initialization: $t_x, t_y \leftarrow \sum_{n=0}^{N-1} \cos \theta^n, \sum_{n=0}^{N-1} \sin \theta^n$
2 $t \sim Uniform(1, ..., T)$
3 $\epsilon \sim \mathcal{N}(0, I)$
4 $\hat{\theta} \leftarrow model(\sqrt{\hat{\alpha}}\theta_0 + \sqrt{1 - \hat{\alpha}_t}\epsilon, (t_x, t_y), t)$
5 $Loss \leftarrow MSE(\hat{\theta}, \theta)$
6 $Dist \leftarrow MSE(\sum_{n=0}^{N-1} \cos \hat{\theta}^n, \sum_{n=0}^{N-1} \sin \hat{\theta}^n)$

Algorithm 2: Pseudo-code of sample IKDP.

Input: A target pair t_x, t_y
Output: The prediction $\hat{\theta}$
1 Initialization: $t \sim Uniform(1, ..., T)$
2 **for** *t in [T, ..., 1]* **do**
3 $\quad z \sim \mathcal{N}(0, I)$ if $t > 1$, else $z = 0$
4 $\quad \theta_{t-1} \leftarrow \frac{1}{\sqrt{\alpha_t}}(\theta_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_t}}\epsilon(\theta_t, t)) + \sigma_t z$
5 **end**
6 Return: θ_0

IV. Experiments

A. Visualization

We have visualized the training process and data sets. The first is to gradually add noise following the Gaussian distribution to the target point. Although in the normal Cartesian coordinate system, the trend of the Gaussian distribution that he gradually becomes cannot be observed. But you can map it back to the $\arccos t_x$ coordinate system, and you will find that it will gradually change from a uniform distribution (because I drew tens of thousands of target coordinates at a time) to
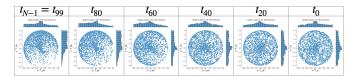

Fig. 6: Unity distribution in different diffusion time steps

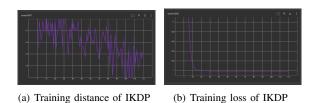(a) Training distance of IKDP (b) Training loss of IKDP

Fig. 7: In the training process of our IKDP, it can be seen that the Loss aspect converges quickly, and then slowly decreases; while the distance part gradually decreases like training FID, but it does not strictly decrease.

a normal distribution. Please see the Fig. 6 from the uniform distribution at t_0 to the normal distribution.

In the training part, we compare the output by the model with the correct angle $(\hat{\theta}, \theta)$, and visualize the distance between the tip and the actual target after the angle is applied to the robot $(\hat{t}, t)$. It can be found that the latter converges quickly because it is used as a loss function for model training. The former is positively correlated with loss, that is, there is a tendency to gradually decrease, just like training an ordinary Diffusion process, FID will not strictly decrease, but the trend will gradually decrease. In the Fig. 7, you can see the values recorded by tensorboard.

Next is to visualize the diffusion process of a certain target. In the figure Fig. 8, the red cross is the given target (t_x, t_y), and each line is the denoise process. What is relatively light is the joint angle combination that has not been denoised. After several rounds of generation, it will gradually become darker, and the darkest is the result of the final diffusion, that is, the generated joint ends $(\hat{t_x}, \hat{t_y})$ correctly point to the target (t_x, t_y).

After designing the model and visualization, we did several experiments.

B. Experiments with different joints

The first is the experiment on the number of different joint points. We did experiments with 2, 4, and 8 joints respectively. It can be found that the more joints, the smaller the loss. For such a result, we speculate that the more joints the higher the fault tolerance rate. In the experiment of two joint points, there is almost only one set of inverse kinematics solutions for each angle, so the error of one joint angle can easily lead to a long distance from the target.

Fig. 8: Pose Estimation Calculation for the depth of joints

Timesteps	Angle Distance↓	Target Distance↓
2	0.499	1.982
4	**0.32**	**1.283**
8	0.243	8.03

TABLE I: For experiments with different joints, we speculate that the more joints the higher the fault tolerance rate. In the experiment of two joint points, there is almost only one set of inverse kinematics solutions for each angle, so the error of one joint angle can easily lead to a long distance from the target.
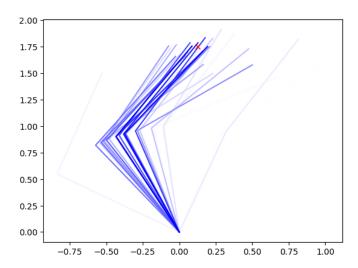
C. Experiments at different timesteps

We think that in this task, unlike DDPM for image generation, the task of generating one-dimensional numbers does not require so many diffusion processes. On the one hand, it can speed up the generation and save unnecessary iterations. After adjusting the parameter settings many times, it can be seen from the experimental results that the highest accuracy rate will be achieved at 80 timesteps, which also verifies the previous speculation. The results are shown in Table II.

D. Compare with other models

Although this topic is not popular, there are indeed other teams doing research here. See the TableIII. Like IKNET [2] proposed in early 2022, or simply MLP. Therefore, we are here to make some comparisons with other people's work, but because of the different methods, they only provide the mean-square error of the distance from the target, without the distance between each joint, so they only compare the former with the execution time. Our work is much better than other machine learning methods, such as IKNET. In terms of execution time, DDPM [1] is slower than pure MLP or IKNET because it takes many iterations to calculate the answer. However, like the nature of DDPM, it has a relatively high generation quality.

Timesteps	Angle Distance↓	Target Distance↓
40	1.382	4.361
80	**1.283**	3.611
100	1.37	**3.44**

TABLE II: For experiments with different timesteps, it can be found that 80 timesteps are more satisfactory in performance, and larger or smaller sizes will not have better preformance.

Methods	Target Distance↓	Runtimes↓
MLP	12.3	0.04
IKNET [2]	2.3	**0.007**
IKDP (Ours)	**1.98**	0.065

TABLE III: Compared with the previous state-of-the-art that used machine learning to find the inverse kinematics solution, it can be seen that the accuracy is much higher than the former, but the execution speed is slightly slower.

E. Future Work

In addition to the currently completed content, we are also advancing our project. First of all, we think that the model still has a lot of information to consider, and only the Transformer Encoder/decoder seems to be a bit insufficient. In addition, as mentioned above, this technology can be actually applied to joint control to determine whether the theoretically feasible model will have serious delay or whether it is accurate. Finally, on the topic of inverse kinematics, in fact, in addition to expecting the end of the robot to reach the given coordinates (x, y, z), it is also hoped that the contact angle of the robot will reach (yaw, pitch, roll). We believe that the model plus other design can accomplish this task.

V. CONCLUSION

In this Final project, we use the DDPM [1] concept to calculate the Inverse kinematic, that is, to generate the angles that each joint of the robot should have under the given target position of the robot tip. At the same time, we changed the ordinary DDPM into a conditional DDPM by adding conditions at the embedding time, so that the model can generate the joints of the robot so that the end touches our given conditions. In terms of generation ability, it is much better than the previous SOTA (IKNET), and the generation speed is also within one second. Finally, through the simplification of the formula, we observe the rotation matrix between the bones and change the formula that was originally calculated separately into an inner product or a parallel addition. The original data generation speed is increased to the millisecond level, and the data set of 100,000 records can be quickly generated.

condition DDPM derived by other teams are also Inspired our design model. Of course, the professors and teaching assistants of this course also gave us valuable opinions and ideas.

REFERENCES

[1] Jonathan Ho, Ajay Jain and Pieter Abbeel. Denoising Diffusion Probabilistic Models, 2020; arXiv:2006.11239.

[2] Raphael Bensadoun, Shir Gur, Nitsan Blau, Tom Shenkar and Lior Wolf. Neural Inverse Kinematics, 2022; arXiv:2205.10837.

[3] Zeeshan Bhatti, Asadullah Shah, Farruh Shahidi and Mostafa Karbasi. Forward and Inverse Kinematics Seamless Matching Using Jacobian, 2014, Sindh University Research Journal (SURJ) Volume 45 (2), 8/2013, pp:387-392, Sindh University Press; arXiv:1401.1488.

[4] Jooyoung Choi, Sungwon Kim, Yonghyun Jeong, Youngjune Gwon and Sungroh Yoon. ILVR: Conditioning Method for Denoising Diffusion Probabilistic Models, 2021; arXiv:2108.02938.