Improving Pose Authoring with Constraint Optimization in ProtoRes: A Combined Learning and Kinematics Approach

Farhan Hussain (fha29@sfu.ca)

April 2023

1 Introduction

In recent years, modeling human poses and learning pose representations have become increasingly important due to their applications in computer graphics, animation, motion estimation, and more. Traditional real-time pose manipulation tools, such as CCD, FABRIK, and FinalIK, rely on non-learnable kinematic models, which often generate unnatural poses from sparse constraints. This final project report will outline the implementation of the ProtoRes architecture, as described in the paper *ProtoRes: Proto-Residual Network for Pose Authoring via Learned Inverse Kinematics*[Ore+21] and extend it with a constraint-based optimization component using the FABRIK algorithm [AL11]. The goal is to develop a more efficient and accurate method for constructing full static human poses from sparse user inputs while satisfying user-defined constraints.

2 Related Works

The traditional methods, such as CCD, FABRIK, and FinalIK, when used alone, have limitations in generating natural human poses from sparse constraints because they lack the inductive bias provided by learning from data. However, when you combine these methods with a data-driven approach like ProtoRes[Ore+22], you can potentially overcome these limitations.

In this project, I have used FABRIK [AL11] as a constraint-based optimization component to enhance the ProtoRes architecture. By adding FABRIK to the existing ProtoRes model, I aim to satisfy user-defined constraints while maintaining the naturalness of the generated poses. This combination takes advantage of both the learning capabilities of ProtoRes and the constraint-solving capabilities of FABRIK to create a more efficient and accurate method for constructing full static human poses from sparse user inputs.

3 ProtoRes Architecture + FABRIK

It is important to understand how the ProtoRes architecture works before working on adding FABRIK to it. Here's a simple explanation of the ProtoRes architecture [Ore+21]:

1. **Inputs:** The user provides sparse inputs, such as the position or orientation of certain body joints. These inputs are called effectors and can be positional, rotational, or look-at effectors. Positional effectors define joint positions, rotational effectors control joint rotations, and look-at effectors help align the orientation of a joint towards a target.
2. **Encoder:** encoder: The encoder processes these effectors for translation invariance and embeds them. It then uses a proto-residual structure to convert the pose information from effectors into a single vector, called the pose embedding.
3. **Decoder:** decoder: The decoder takes the pose embedding and expands it into a full-body pose representation, including the local rotation and global position of each joint.

There are two ways to incorporate FABRIK with ProtoRes:

1. Directly incorporate FABRIK into ProtoRes, this could be done by embedding FABRIK into the decoder of ProtoRes.
2. Integrate FABRIK as a post-processing step: After training ProtoRes, we can use the generated pose as the initial pose for the FABRIK algorithm [Ari]. The input to FABRIK will be the joint positions generated by ProtoRes, and the output will be a pose that satisfies the user-defined constraints.

I went with the 2nd approach, this was done as the first approach would have required significant changes to the initial ProtoRes architecture, from cleaning the dataset to the output of the generated poses.

4 Algorithm/Method

My project implements the ProtoRes architecture and extends it with a constraint-based optimization component using the FABRIK algorithm. The steps involved in implementing the algorithm are:

1. Review the paper thoroughly and understand the architecture and methodology.
2. Acquire the necessary datasets based on high-quality human motion capture data.
3. Implement the ProtoRes architecture using a suitable deep learning framework, such as TensorFlow or PyTorch.
4. Implement the FABRIK constraint solver to enforce user-defined constraints during pose generation.
5. Train the model using the provided/custom datasets.
6. Develop a user interface for manipulating the effectors (positional, rotational, and look-at) and integrating the model into a real-time 3D development platform, such as Unity.
7. Test the model on various skeletons and poses

The code for the initial implementation of ProtoRes is taken from the ProtoRes paper's Github page [Uni]. I made changes to the ProtoRes model so that it will be compatible for me to work on, these changes are in the *changeModel.py* file. This file makes it compatible with Unity's Barracuda framework. *CharacterPoseController.cs* allows pose manipulation and *SimpleFABRIK.cs* allows to integrate the FABRIK algorithm.

4.1 Unity integeration

One of the major challenges was integrating ProtoRes with Unity as the original paper does not provide any way to integrate the output of ProtoRes with Unity. My assumption is that even though the paper is published by the Unity Labs, they might integrate ProtoRes directly in a future update to the Unity platform.

To address this issue, I used the Barracuda framework. The model file was not initially compatible with Barracuda so I had to make changes to the onnx file. To do this I used Netron [23] to investigate which layer was causing issues. The issue was related to the "if" layer (figure 1), I converted it to the "where" layer (figure 2).

5 Results/Evaluation

The paper did not provide a model file to easily compare the results to. Therefore I trained on the dataset provided (miniMaximo and miniUnity). Due to compute limitations (trained on a 1650 TI GPU), I could not train on the hyperparameters provided in the paper as it would take well over 200+ hours to train. I fine-tuned the parameters and the training time was reduced to around 3 hours. As the final results would not be comparable to that of the paper, I decided to not evaluate based on the F1 score as provided in the original paper [Ore+21]. Instead the evaluation will be based on what was set forth in the original proposal.

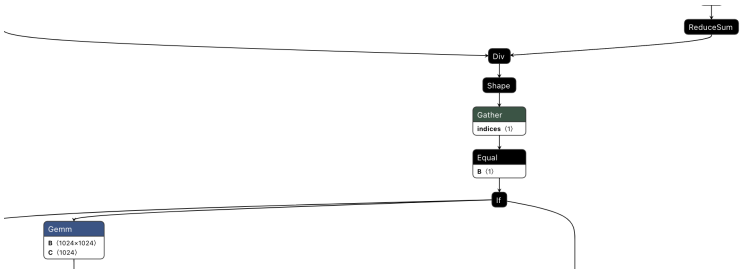

Figure 1: If layer

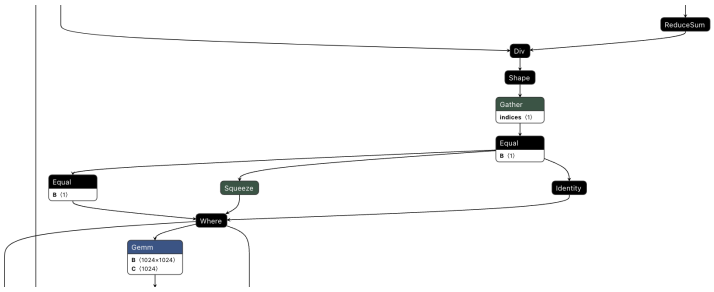

Figure 2: Where layer

1. A functioning implementation of the extended ProtoRes architecture capable of generating accurate and natural human poses from sparse user inputs while satisfying user-defined constraints. (**Implemented**)
2. A user-friendly interface for manipulating the effectors and integrating the model into a real-time 3D development platform. (**Implemented**)
3. Demonstrations of the model's versatility in handling various skeletons and poses, including retargeting and retraining for different skeletons and quadruped datasets. (**Not Implemented** due to time constraints)

The current version performs poorly as can be seen in the demo, this is a result of three things. Firstly it is the way in which FABRIK was implemented along with ProtoRes, it might not have been the most efficient way. Secondly I was not able to train the ProtoRes model as much as the original paper did (it would take 200+ hours). Thirdly, creating a UI to implement ProtoRes in Unity was a challenge, so these are some possible reasons why.

6 Conclusion

There were significant challenges during this project. From understanding the complicated architecture of ProtoRes to acquiring powerful GPUs for training and finally integrating it with Unity. The biggest challenge was integrating with Unity as there was no native way to incorporate it with Unity. Also since the model I trained was different from the original paper's (reduced training time, different hyperparameters), in the future I would like to fully utlize powerful GPU's so the final result will look similar to the one generated in the paper. Also I would like to incorporate ProtoRes on other quadruped datasets.

7 Acknowledgements

I would like to thank the Unity Labs for providing the initial code for the implementation of ProtoRes [Ore+21] [Uni]. [Ari] [AL11] for FABRIK algorithm.

References

[AL11] Andreas Aristidou and Joan Lasenby. "FABRIK: A fast, iterative solver for the Inverse Kinematics problem". In: *Graphical Models* 73 (Sept. 2011), pp. 243–260. DOI: 10.1016/j.gmod.2011.05.003.

[Ore+21] Boris N. Oreshkin et al. *ProtoRes: Proto-Residual Network for Pose Authoring via Learned Inverse Kinematics*. 2021. DOI: 10.48550/ARXIV.2106.01981. URL: https://arxiv.org/abs/2106.01981.

[Ore+22] Boris N. Oreshkin et al. "ProtoRes: Proto-Residual Network for Pose Authoring via Learned Inverse Kinematics". In: *International Conference on Learning Representations*. 2022.

[23] Apr. 2023. URL: https://netron.app/.

[Ari] Andreas Aristidou. *Forward And Backward Reaching Inverse Kinematics*. URL: http://andreasaristidou.com/FABRIK.html.

[Seb] Sebastianstarke. *Sebastianstarke/AI4Animation: Bringing characters to life with computer brains in Unity*. URL: https://github.com/sebastianstarke/AI4Animation.

[Uni] Labs Unity-Technologies. *Labs/Projects/protores/code at Main · Unity-Technologies/Labs*. URL: https://github.com/Unity-Technologies/Labs/tree/main/Projects/ProtoRes/code.

[Ore+21] [Ore+22] [Seb] [AL11]