The re-implementation project of Simbicon

Duan Min, Zhiyi Xu @ University of California, Davis

March 2019

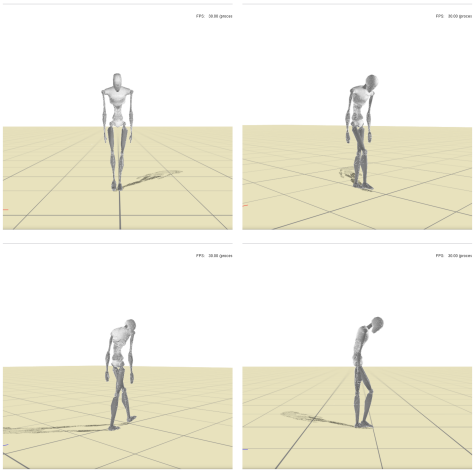

Figure 1: Simbicon C++

Abstract

Simbicon is a simple control strategy that can be used to generate a large variety of gaits and styles in real-time.[4] The examples that are given including all directions walking, running, skipping, and hopping. After studying the paper and the code, we decided to re-implement some important controlling part of the program and expand a JavaScript re-implement version, in order to better understand and present this paper. This is the final project for ECS 279 at UC Davis for Min Duan and Zhiyi Xu.

1 Introduction

In short, Simbicon will generate a simulated character and allows user to control the movement in some ways of the character. More details and discussion about the algorithms and strategies of Simbicon will be included in the next section (Related work). On the website of Simbicon includes a C++ version for 3D simulator and a Java version for 2D simulator. We also found a simplified JavaScript version of Simbicon on GitHub.[2] For some unknown reasons, the Java version is not working. Since our team does not major in Java, we decided to skip this part. For this final project, we are re-implementing the central controlling part of the Simbicon C++ version. However, the re-implementation might not affect the output animation of the C++ version. So, we also decided to extend the simplified JavaScript version to present more various outputs and styles of working in this way.

2 Related Work

2.1 C++ Version

Given that we did not have any prior experiences with programming simulation applications and lack of background knowledge in physics, we had to dive into the details of the paper, and further investigate the formulas that drives the character to the desired pose denoted by the according state in the finite state machine.

To learn the implementation of a walking controller, we started by learning the major components of the controller proposed by paper. The overview of the

framework is quiet straight-forward. The user provides character to render and controller to use as an input file. The system will load all the necessary parameters for simulation, such as the pose information for each state. The walking style is dictate by a finite state machine, which contains the target pose for each phase in a walking cycle. The "Driver" that drives joints to their designated positions is a function called `PoseController::computeTorques()`. In the **Balance Control Strategy** 3.1 section of the paper, the authors present the following formula to calculate the torque to be applied for a joint.

$$\tau = k_p(\theta_d - \theta) - k_d\dot{\theta}$$

where k_p is the proportional gain, and k_d is the derivative gain of the Proportional-Derivative controller that Simbicon is using to derive the feedback.

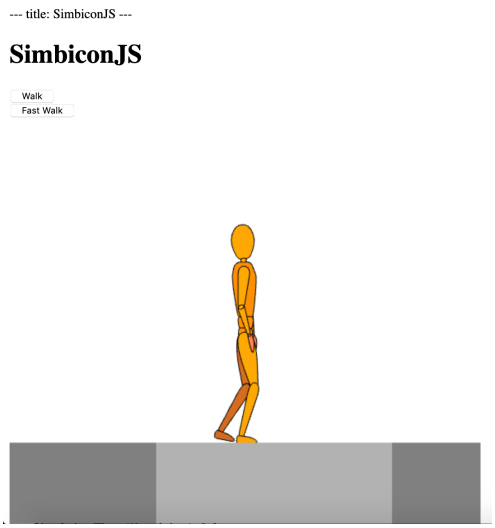

Figure 2: SimbiconJS

2.2 JavaScript Version

The JavaScript version (Fig. 3) is implemented by Michael Firmin on GitHub. [2] It only includes a normal walk and a fast walk in the program, and the simulation is 2D. The project contains 4 different parts that each has a JavaScript file for it. The coach.js (COntrolling Articulated CHaracters) is an all-in-one simulation and rendering framework for controlling the motion of dynamically simulated rigid articulated characters [3], which is also developed by Michael Firmin. The human.js defined the character and mesh.js contains all the vertices and color. At last, the simbicon.js is the main part of the project, which initialize everything and make the character move according to the style selected by user. The underlying physics engine is Ammo.js [5], which is the JavaScript version of Bullet [1].

3 Implementation

3.1 C++ Version

Our goal is then to understand the above formula that is mentioned in previous section and re-implemented on top of Simbicon's framework. To do this, we first need to study the structure of Simbicon and figure out its control flow. Below is a diagram of the execution flow of how the simulation process starts.

The code we re-implemented is in `PoseController.cpp` from `line 133` to `line 241`, a method of class `PoseController` called `newComputeTorques()`. Reading through Simbicon's implementation, we noticed that Simbicon use quaternion to represent relative orientations of joints. From our understanding of the paper and the code, We infer that Simbicon uses quaternion because it often requires transformation between joints' local coordinate system, and world coordinates. Using quaternion avoid issues such as gimbal lock as well as makes transformations between frames a simple one liner `q1.rotate(v)`, given that `q1` is a quaternion and `v` is the vector to rotate. In this case, to re-implement the calculation of torques on top of Simbicon's existing framework, we would have to use quaternions. We did some study to understand the conversion from quaternions to Euler angle, and how does rotation using quaternion carry out, which helps us completely comprehend `PoseController::computePDTorques()` and re-implemented it. We provide really detailed

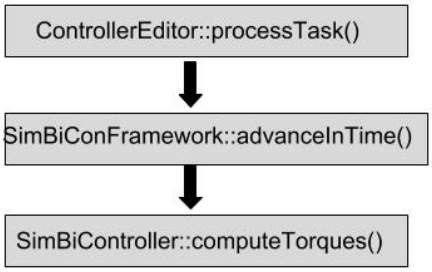

Figure 3: Execution Flow

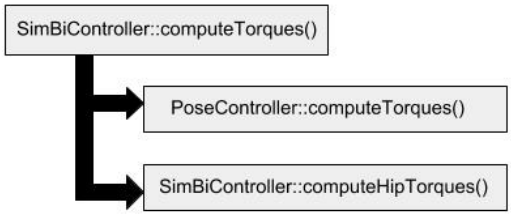

Figure 4: Torque Calculation

explanation of the derivation in the comment section

Another core part of Simbicon's control strategy is torso and swing-hip control, as mentioned in the paper. Below is a diagram showing the overview of how the torque is calculated at every stage of the simulation.

The according implementation is in a method of `class SimbiController` called `computeHipTorques()`. The relationship between torso and two hips is characterized by following formula.

$$\tau_A = -\tau_{torso} - \tau_B$$

. To help the torso, which is referred as "root", in the code, realize the desired torso orientation, we need to apply extra forces on the hips such that besides providing torques derived using PD controller, it exerts a net torque on the torso. Compared to the PD controller, the calculation is quite straight forward. We were confused about the damping factor and predictive torque appeared

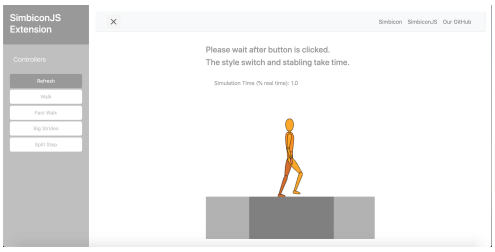

Figure 5: Extended JavaScript Version

in the original implementation, Since the paper did not cover the details about them. Thus, we removed related lines of code to the impact on the simulation. From what we tested, we could not see any difference in the walking without disturbation of sudden push. One of the things we can further investigate is that whether removing those two terms affect pose recovering when given a sudden push. We re-implemented a method called `SimBiController::newComputeHipTorques()` in file `SimBiController.cpp` from `line 464` to `line 545`.

3.2 JavaScript Version

For this project, we are only focusing on the extending the different styles of moving, which is controlled by the Simbicon parameters. There are 13 parameters in total, which covers delta time, balance feedback, torso, swing hip, swing knee, stance knee, and ankle. However, when shifting from one style to another style, the character went very unstable and lose the balance. As a result, we decided to make the character switch to normal walk style for a few seconds and then move on to the style that user chose.

```
setSimbiconParameters({
    dt: .3,
    cde: 0,
    cdo: -2.2,
    cve: -.2,
    cvo: 0,
    tor: 0.,
    swhe: -.4,
```

```
    swho: .7,
    swke:  1.1,
    swko: .05,
    stke: .05,
    stko: .1,
    ankle: -.2,
});
```

Listing 1: Parameters for Walk

We also add some basic features to the website to make it more user friendly. (Fig. 4) We add a control panel to the left and put all the controlling buttons there. The panel is collapsible so that all the space can be given to the animation. The upper right corner includes all the links for related work.

4 Conclusion

For this project, we studied the Simbicon, which is a simple control strategy that can be used to generate different gaits and styles in real-time. We especially focused on the C++ version of the Simbicon and re-implement the functions that compute the torques applied at every stage of the simulation to drive the character to achieve desired pose, which involves deriving the torque using PD controller, as well as calibrating the torque on two hips. We also found a JavaScript version of the Simbicon on GitHub. We studied it and extended the function of it.

5 Future Work

For the re-implement of Simbicon, it will be the best if we can re-implement all the part of Simbicon to have a more comprehensive and clear understand of the Simbicon. For the extension of the JavaScript version, the current work only includes some style of walking. To improve the current work, the project should allow user to change the character and include some modification for the upper body too. Moreover, the current version is 2D, for future work, it can be extended to 3D either.

References

[1] Bullet Physics development team. Bullet physics sdk, 2018.

[2] Michael Firmin. Simbiconjs, 2016. Created in Oct 2015.

[3] Michael Firmin. coach.js, 2017. Created in Dec 2015.

[4] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: Simple biped locomotion control. *ACM Trans. Graph.*, 26(3):Article 105, 2007.

[5] Alon Zakai et al. ammo.js, 2019. Created in May 2011.