Apply KUO FeedForward Model to Simulated Robotic System.

# Introduction

This project attempted to model a control system for rhythmic movements based on the controllers described in Kuo et. al. Our goal was to augment the feedforward controller described in the paper with an optimal controller using trep and SAC(Sequential Action Control). Our prediction was that SAC would model a controller that we expected would behave more efficiently compared to the hybrid feedforward and feedback model described in the paper. We found that contrary to our initial belief, applying Kuo's feedforward model to a trep system was not as simple as replicating the equations in code. We soon realised that the Kuo model was very simplified and idealized in a way that using trep's dynamic simulation does not allow. Additionally, the model described in the paper is not well suited for use with SAC, since the velocity equations have points of singularity.

# Real-world Implementation

Trep is designed to simulate systems much closer to a “real” robotics system. Although Kuo’s paper implicitly claims to be suitable for robotic implementation, we propose that as it currently stands, the dynamics are not wholly practical. We initially solved Kuo's feedforward model in both mathematica and python, where we could use equations of motion directly out of the paper, where the mass and kinematics need not be explicitly defined. Both programs gave satisfactory results identical to the paper. Once we attempted to construct the Kuo model in trep, we were able to gain a better understanding of the equations and model. We realized that Kuo's implementation and our mathematica and python simulation effectively halts the integration of the dynamics, to instantaneously apply an impulse control and then restart the integration with the new initial conditions. We quickly realized that this was not possible in trep and not realistic for a robotic system. Additionally, Kuo's model was dimensionless. Real world systems are not dimensionless, and trep explicitly requires system dimensions such as mass to be defined. For our implementation we used dimensions comparable to that of a human leg. This resulted in special considerations for issues of discrete time and impulse that were not explicitly mentioned in the Kuo paper, and were also not an issue in the python and Mathematica implementations.

# Discrete Time

In order to apply the impulse we had to sample the time and write a control law that applied a very high torque at the peaks of the swinging motion. This took much trial and error in describing the tolerance of how close to the specified time the simulation needed to be. We chose to reduce the window such that the control was applied during only one sampling period. This was our 'slop'-timal solution to this issue, but we recognize that other design choices may have also worked.

# Impulse

We then turned to the impulse itself. We began with an approximation of what the impulse ought to be based on the velocity of the pendulum, the desired velocity, and the application time of the torque. This starting point was around 600N-m of torque, but was not sufficient to create the trajectory found in Kuo. The torque impulse value was increased to 3195 N-m to follow the trajectory from the Kuo paper. This value was very large, and suggested to us that this model of impulsive control signals was too simplified for an actual system.

# SAC

We were unable to implement Sequential Action Control in the way that we intended. Restricting ourselves to using the SAC library restricted the cost functions that we could use to implement a hybrid model, but we did not realize the restrictions it also put on our desired trajectories. The sactrep python library is a wrapper around a C++ library, because of this the desired trajectory function needed to be specified in a particular format. We initially tried to implement a reference to a list generated from feedforward control and a sort of piecewise trajectory. Both raised errors with the library. We could only give continuous time trajectories that were strictly a function of time, i.e. sin[t]. We attempted to use regression methods to fit a continuous time function to our trajectory, but were unable to fit the desired trajectory to any function except a triangular wave. However, again the use of the library wrapper threw an error for this trajectory specification. In order to implement sequential action control that tracks the trajectory from the Kuo paper, we would need to code SAC ourselves in python rather than using the still in development sactrep library.

# Conclusions

Although we were unable to successfully execute the hybrid control system that we had envisioned, insight was gained in our implementation of the feedforward Kuo system. First, we realized that the model was unrealistic for any application to a real-world system, due to its dimensionless generalization, instantaneous change in velocity, and assumption of continuous time feedback. For these reasons, the feedforward model was difficult to implement in a dimensioned, discretely sampled simulation in trep. After successfully implementing the feedforward model, time constraints and some deficits of expertise prevented us from implementing SAC as we had hoped.