UKRAINIAN CATHOLIC UNIVERSITY

MASTER THESIS

Cloud-based just-in-time computation of dynamic transformations of biomechanical models

Author:
Dmytro MYKHAILOV

Supervisor:
Dr. Sergiy YAKOVENKO

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2024

Declaration of Authorship

I, Dmytro MYKHAILOV, declare that this thesis titled, "Cloud-based just-in-time computation of dynamic transformations of biomechanical models" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"The best mistake is the one that is made during training."

Hryhorii Skovoroda

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Master of Science

Cloud-based just-in-time computation of dynamic transformations of biomechanical models

by Dmytro MYKHAILOV

Abstract

To manage the movement control of the human body or robotic systems, the problem of calculating dynamic transformations must be solved. Solving the dynamic transformation problem in real-time can be a complex task due to the complex human structure with a large number of degrees of freedom. To solve such problems, a large amount of computing resources or the use of artificial neural networks, which must be created for each model separately, are required. The following study presents a cloud-based solution for the problem of real-time dynamics transformation with the ability to collect the processing data. Such a solution will help solve the problem of limited computing and energy resources on edge devices, as well as systematize and collect data for subsequent research. The proposed solution faces the task of computing dynamic transformation in real time with the conditions of network delays. This master thesis proposes to extrapolate previously obtained data by dead reckoning the simulated model using Unscented Kalman Filter.

Acknowledgements

I extend my deepest gratitude to my project advisor, Sergiy Yakovenko, whose guidance, expertise, and unwavering support have been invaluable throughout the journey of this research. His insightful feedback and encouragement propelled this thesis forward.

I would like to express my sincere appreciation to Joost B. Wagenaar and Edmore Moyo from the University of Pennsylvania for their technical assistance and collaboration. Their expertise and contributions played a pivotal role in shaping the development and implementation of the proposed framework.

Furthermore, I am indebted to the Ukrainian Catholic University, particularly the Faculty of Applied Sciences, for providing the conducive academic environment and resources essential for conducting this research.

Lastly, I would like to extend my heartfelt gratitude to the Armed Forces of Ukraine for their unwavering dedication to protecting our nation and ensuring our safety. Their sacrifices and commitment to duty serve as a constant source of inspiration and motivation.

To all those mentioned above and to countless others who have supported me along the way, I am profoundly grateful for your encouragement, guidance, and unwavering belief in my abilities. This thesis is as much a reflection of your contributions as it is of my own efforts. Thank you!

Contents

List of Figures

List of Tables

List of Abbreviations

SotA	**S**tate-**o**f-**t**he-**A**rt
ANN	**A**rtificial **N**eural **N**etworks
dof	**d**egree-**o**f-freedom
UKF	**U**nscented **K**alman **F**ilter
EKF	**E**xtended **K**alman **F**ilters
NEES	**N**ormalized **E**stimated **E**rror **S**quared

Dedicated to my family and friends.

Chapter 1

Introduction

1.1 Motivation

Nowadays, one of the prior tasks of neuroscientists and physiatrists is the development of prosthetics with human-machine interaction support. The successful development of such advanced prosthetics can significantly enhance the quality of life of people with disabilities caused by different traumas or aging. Due to the complex human structure described by Winter, 2009, the real-time simulation of biomechanics of human movement is considered to be a challenging task.

To develop advanced prosthetics, it is necessary to solve the problem of decoding limb control signals followed by the computation of limb dynamics. This task includes processing nonlinear and high-dimensional neural and mechanical signals. To solve the speed-accuracy tradeoff for the calculation of limb dynamics, researchers have focused on applying both classical numerical methods of approximation described by Featherstone, 2008 and state-of-the-art (SotA) methods with the use of artificial neural networks (ANN) researched by Manukian, Bahdasariants, and Yakovenko, 2023.

Kinematics and dynamics are two branches of mechanics in physics that study the movement of bodies. Kinematics describes motion without considering what causes that motion, while dynamics looks at the forces and moments that cause motion.

Kinematics studies ways of describing the movement of bodies, such as trajectories, velocities, and accelerations, without considering the forces and moments that cause movement. In robotics and biomechanics, kinematics describes the movement of joints and links of robots or body parts.

At the same time, dynamics studies the reasons for the movement of bodies, i.e., forces and moments causing this movement. In robotics and biomechanics, forward and inverse dynamics problems are crucial in understanding and controlling movement.

The task of forward dynamics is to determine the motion of a system (position, speed, acceleration) under known forces and moments. In this problem, the initial conditions (position and velocity) and applied forces are known, but we need to determine how the system will move in time.

The task of inverse dynamics is to determine the forces and moments necessary to achieve the system's given motion (position, speed, acceleration). In this problem, the trajectories of movement (usually positions and velocities) are known, but it is necessary to determine the forces and moments that cause this movement.

In biomechanics, these problems are used to analyze human and animal movements. Forward dynamics helps to understand how muscles and external forces influence body movement, which is essential for sports training and rehabilitation. Inverse dynamics is used to estimate forces in joints and muscles when performing

movements, which is important for developing orthopedic devices and analyzing movement patterns.

At the current network coverage and bandwidth level, an alternative option would be moving the computations to the cloud. This approach has proven itself with great results for robot control. The use of clouds also could potentially seriously expand the possibilities of neuroscience. SotA research in the Cloud Robotics field made it possible to increase computational power, gain access to big data, and standardize data storage and processing, which saves energy and physical space and reduces processing delays, according to Saha and Dasgupta, 2018.

Adapting the achievements in Cloud Robotics would make it possible to reduce the sizes of advanced prosthetics and increase their autonomy, bringing a new impetus to prosthetics. Additionally, cloud computing will also be able to expand the functionality of wearable smart-health solutions and expand the possibilities in the fields of rehabilitation, sports, or general health.

1.2 Thesis Structure

The following position paper holds the following structure:

- Chapter 2, "Related Work", describes current SotA methods for computing real-time limb dynamics, fundamental research in the Cloud Robotics field, and distributed simulations.
- Chapter 3, "Problem Setting and Approach to Solution", sets the research questions, proposes research hypotheses, and forms a particular approach to solve problems.
- Chapter 4, "Experiments", includes a description of the methodology, setting up experiments, conducting research, analyzing the data obtained, and discussing the results in the context of the hypotheses.
- Chapter 5, “Conclusions,” includes a summary analysis of the study, a discussion of the significance of the findings, and ideas for further research.

Chapter 2

Related Work

2.1 Review of Related Work

2.1.1 OpenSim Package

The software package OpenSim, first introduced by Delp et al., 2007 and its current ability presented by Seth et al., 2018, has now become one of the main tools in biomechanics and rehabilitation research. This open-source package includes all the necessary components for simulating musculoskeletal dynamics and neuromuscular control. It allows the creation and editing of biomechanical models, simulation of musculoskeletal dynamics and neuromuscular control, and visualization of results. OpenSim computes the dynamics of multibody systems using an order-N recursive formulation and is not initially designed to simulate real-time dynamics. Despite that, Pizzolato et al., 2016 were able to develop a real-time inverse kinematics and inverse dynamics solver for the lower limb using OpenSim. For lower limb models with three-dimensional and 23-degree-of-freedom (dof), they were able to achieve inverse kinematics and dynamics calculations without simplifications at 2000 frames per second with less than 31.5 milliseconds of delay. This was achieved by taking advantage of multi-threaded data processing with inverse kinematics and inverse dynamics calculations on different threads. The mentioned experiments were conducted using a Dell Precision Workstation T7500, with 2 Intel® Xeon® Processors X5660 (12MB Cache, 2.80 GHz, 6 cores) and 8GB of RAM.

2.1.2 MuJoCo Physics Engine

An alternative tool for the simulation of biomechanical data is the MuJoCo physics engine developed by Todorov, Erez, and Tassa, 2012. In comparison with OpenSim, MuJoCo is not a tool created for biomechanics specifically, but it is a general physics engine for modeling multi-joint dynamics with contact. MuJoCo also provides the ability to create and modify physical models and simulate model dynamics. The implementation of MuJoCo is based on numerical optimization since numerical optimization is the most powerful and generally applicable tool for automating processes by engine creators. The multi-thread out-of-the-box implementation is the main feature of MuJoCo. Erez and Todorov, 2012 by using the 15-millisecond time-step, were able to achieve the results 500 times faster than during the real-time for the inverse dynamics computing (without inverse kinematic) for the 31-dof simulated humanoid. Todorov, 2014 also reported that with the use of a time-step of 10-millisecond, he was able to achieve the results 100 times faster than during real-time (one cycle per 0.1 milliseconds) for calculating forward dynamics for the 27-dof humanoid with ten contacts. The mentioned results were obtained with the use of an Intel® Xeon® X5860 processor (12MB Cache, 3.33 GHz, 6 cores).

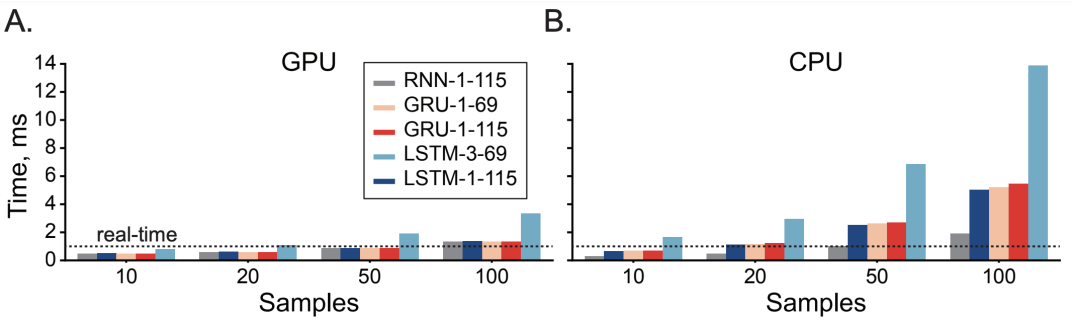

FIGURE 2.1: Average forward propagation latencies as a function of sequence length. From Manukian, Bahdasariants, and Yakovenko, 2023.

2.1.3 Unscented Kalman Filters for Real-Time State Estimation

Lowrey, Dao, and Todorov, 2016 presented a method for the estimation of whole-body dynamics state in real-time. Darwin-OP humanoid biped robot with a 26-dof was used for the experiment. The calculations were performed on a laptop with a Core-i7 4710HQ processor. Under these conditions, a complete update cycle takes seven milliseconds. For this purpose, the Unscented Kalman Filter (UKF) with a uniform weighting scheme was developed. Applying uniform weighting enhances resilience against samples that defy contact constraints. One of the critical points of efficiency is also the calculation of forward dynamics with the MuJoCo physics engine, which effectively uses the capabilities of multithreading.

2.1.4 Real-Time Inverse Dynamics for Arm and Hand Using ANN

Another possible option for computing biomechanical dynamics is the use of ANN networks. For example, ANN guided by signal processing in the human brain-computer interface was used to decode arm dynamics in Manukian, Bahdasariants, and Yakovenko, 2023. In the mentioned work, for a realistic musculoskeletal hand and arm model consisting of 23-dof, ANNs were faster than real-time, but this varied depending on the number of ANN layers and the computing device used (CPU or GPU).The results can be seen on the Fig. 2.1.

2.1.5 Distributed Interactive Simulations in Game Development

A similar problem is solved by distributed interactive simulation techniques during the development of networked games. The main complexities in the development of such systems are shared state and its synchronization. This is due to the nature of the network communication – a shared state from a remote device is always available with a delay. To synchronize shares data, the following types of lag compensation techniques are used:

- The dead reckoning technique (see Fig. 2.2) involves the exchange of information about the position of an object, its speed, and acceleration at a specific moment in time between remote simulations described in "IEEE Standard for Distributed Interactive Simulation (DIS) – Communication Services and Profiles - Redline" 1996. Next, with the use of this information, it is possible to calculate the expected location of a remote object. Globally synchronized clocks between remote simulations are an essential aspect for improving the accuracy of position prediction demonstrated by Yahyavi et al., 2013.

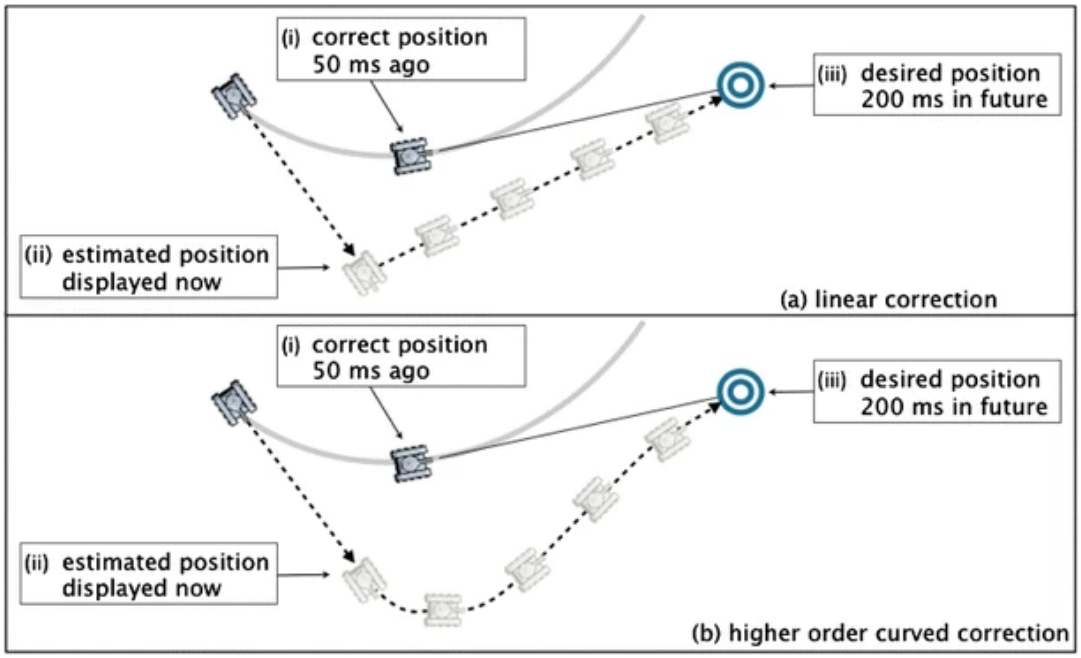

FIGURE 2.2: Corrections to incorrect states predicted by dead reckoning. From Savery and Graham, 2012.

- The local perception filter, presented by Sharkey, Ryan, and Roberts, 2002, and remote lag, proposed by Bernier, 2001, techniques are based on the use of constant lag to the actions of other clients in local simulation, but not for their own (see Fig. 2.3). Authoritative server arbitrates decisions based on the local states of clients. Thus, each client has its own representation of the state, and the server must be able to rewind time to restore the local state (perception) of the client to check actions. The main idea is by exploiting the human perceptual limitations to hide the communication delays.

- The bucket synchronization technique demonstrated by Bettner and Terrano, 2001 consists of slowing down the execution of commands (actions), combining them into batches, and continuing the simulation after synchronizing all states.

- The local lag technique is about slowing down all commands (actions) for a selected amount of time was explained by Mauve et al., 2004. Thus, the local lag of command execution will allow us to cover the network costs and apply them simultaneously to all clients (assuming globally synchronized clocks). The main difficulty is choosing the local lag size so that it is longer than the required time for data delivery but does not significantly affect the responsiveness.

Thus, Savery and Graham, 2012 distinguish the following categories of algorithms for lag compensation:

- Predictive techniques - dead reckoning.

- Delayed input techniques - bucket synchronization, local lag.

- Time-offsetting techniques - remote lag, local perception filters.

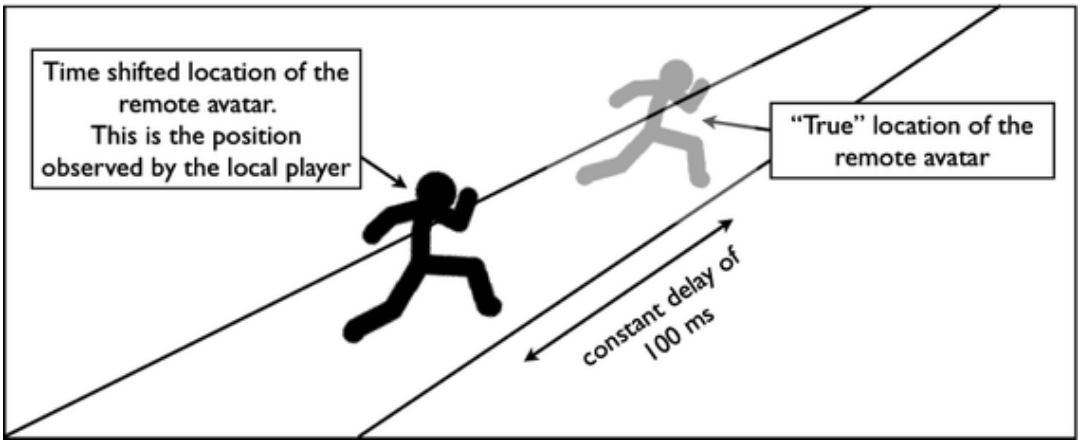

FIGURE 2.3: Time-offsetting technique. From Savery and Graham, 2012.

2.2 Research Gap

As shown, achieving real-time speed when calculating biomechanical data is possible. In the presented works, the real-time speed was achieved by the use of large computing resources or with the use of machine learning approximate solutions. However, placing such hardware, as well as providing the power it requires, into prosthetics or wearable smart health devices is not possible. A possible solution to the described problem could potentially be to move calculations to the cloud, where such hardware could be placed. The move to the cloud could also make it possible to create a data warehouse for all the obtained biomechanical data for the further development of machine learning solutions. In addition to all the obvious advantages of such a model, difficulties with network communication arise. Since communication over the network always has latency or even frame loss, the cloud simulation would constantly lag behind real-time.

Cloud Robotics could be a possible solution to such a problem since it is a popular solution to simplify and minimize the cost of a robot. Most probably, due to the smaller requirements of robotics for the size and computing resources, in comparison to wearable devices, Cloud Robotics is not used for calculating dynamics problems. Nothing other than general rules for minimizing odometry sizes to reduce network latency was found.

Fortunately, the problem of distributed simulations and state synchronization is a common problem in networked game development. In this area, there are a number of lag compensation techniques to solve the problem of networking latency and the variance of latency over time. However, the categories of time-offsetting and delayed input techniques are not applicable since there is no task of network communications between several biomechanical models. At the same time, the dead reckoning approach for lag compensation could suit our problem well. Still, the way with which to build dead reckoning remains an open question since the classic absolute position extrapolation with polynomials option can cause a significant error for complex biomechanical models. For this, the use of Kalman Filters would be a good choice since the use of Kalman Filters for the problems of dynamic estimation in real time has already been showcased by Lowrey, Dao, and Todorov, 2016.

Chapter 3

Problem Setting and Approach to Solution

3.1 Problem Setting

Developing the framework for dynamic transformations of biomechanical models is challenging yet achievable. Producing simulations of biomechanical models in real-time is also accomplishable, with different ways to achieve it being very appreciated. When the mentioned tasks are advanced to the cloud, things change dramatically.

Performing calculations in the cloud means we have to deal with network communications. If the simulation rate in the cloud exceeds latency time, which may occur due to instability in network communication, the simulation will be blocked until the missing packets are received, thus violating the real-time condition. It is significantly aggravated by the simple fact that biomedical signals are taken from complex biomechanical structures, and high-volume data represent them. At the same time, data is often collected at high rates, so the volumes of collected data are pretty large, which undoubtedly will affect the latency of the network. A possible option is to extrapolate previous data by dead-reckoning the simulated model analogously with the game development. Consequently, the first problem is to develop a short-term state estimation method for biomechanical models.

Another challenge that is resolvable lies in the use of cloud resources to the fullest. The possibility of highly parallel computing characterizes cloud resources. Since the goal is to achieve real-time computing, the capabilities mentioned before are thus to be benefited from. Accordingly, choosing a solution that allows you to parallelize calculations and simultaneously be flexible enough to describe the data pipeline is necessary.

3.2 Physics Engine and Models

An integral part of the framework that is under development is a Physics Engine. Physics Engine can be defined as software designed to simulate the physical interactions between objects in the virtual space. Correctly selected Physic Engine makes it possible to create authentic computer simulations to help understand the physical principles behind the movement process and develop SotA technologies and methods in the biomechanics field. Hence, making use of the Physic Engine comes to be a crucial element in the development of the framework for dynamic transformations of biomechanical models.

After thoroughly analyzing the physics engines currently actively used in biomechanics, I eventually chose the MuJoCo. MuJoCo is a freely available general-purpose Physics Engine engineered for robotics and biomechanics. Successful simulations in

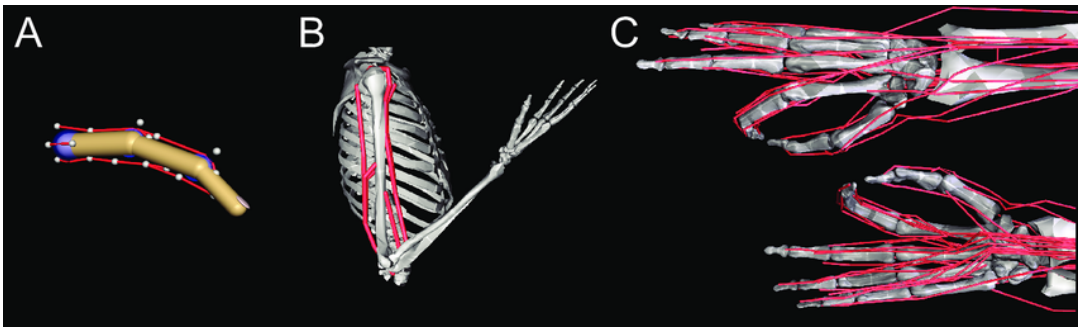

FIGURE 3.1: Musculoskeletal models included in MyoSuite. A - MyoFinger (4 joints - 5 muscles), B - MyoElbow: (1 joint - 6 muscles), C - MyoHand: (23 joints - 39 muscles). From Vittorio et al., 2022.

real-time for the humanoid model, demonstrated by Lowrey, Dao, and Todorov, 2016, were a crucial factor that made me choose the mentioned Physic Engine. An additional factor for such a decision was that MuJoCo was initially developed as a multi-threading solution, making it possible to utilize the cloud resources fully.

MuJoCo offers an extensive specter of capabilities for modeling contact interactions. It allows the creation of models of physical contact between objects, such as friction, adhesion, and reaction force. Such ability is crucial in biomechanics, where interactions with contact play a vital role in the movement of live species and their interactions with their environment.

Furthermore, MuJoCo provides a flexible and easily operated API that allows its users to create, customize, and manipulate models of biomechanical systems without difficulty, as well as integrate such models with other tools and technologies. At the same time, MuJoCo is compatible with operational systems such as Windows, Linux, and macOS, which makes it, together with the developed framework, accessible to a wide circle of researchers and developers in the biomechanics field.

A pivotal aspect for subsequent experiments is the availability of the MyoSuite ecosystem, presented by Vittorio et al., 2022, that is compatible with the chosen MuJoCo physics engine. MyoSuite is a collection of musculoskeletal models, tasks, and agents that are already available for such tasks simulated with the MuJoCo physics engine. Thus, it will allow to conduct experiments not just with the passive dynamic models of human limbs but also with the all-set agents for simulating experiments with the control signal.

MyoSuite consists of five different models: myoFinger, myoElbow, myoHand, myoArm, and myoLeg (see Fig. 3.1). These include simple models, like myoElbow, that contain 2-dof and 6 muscles-tendon units, and complex models, such as myoArm, that contain 2-dof and 63 muscles-tendon units. Different tasks, from simple reaching movements ones to those that include contact-rich movements with object manipulation, are developed for the mentioned models.

3.3 State Estimation

Due to the complex structure of biomechanical models and nonlinear dependencies, for example, due to contact phenomena, it is quite challenging to estimate the state of the biomechanical model. For this problem, it proposed to consider the ability to estimate the state of the model using Kalman Filters, specifically Extended Kalman Filter (EKF) and UKF, since EKF and UKF are good ways to estimate nonlinear state,

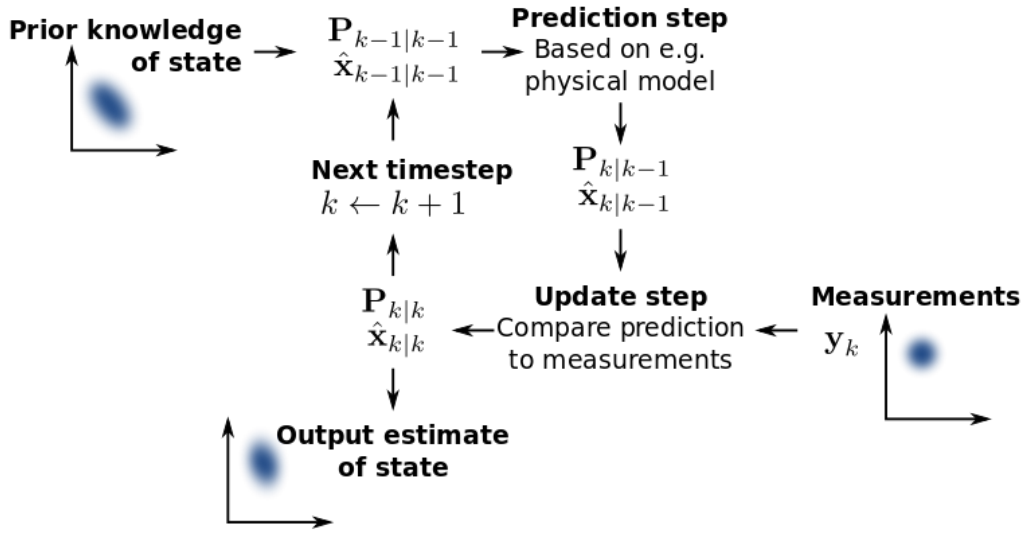

FIGURE 3.2: Basic steps of Kalman filtering. From Aimonen, 2011.

and also in the work of Lowrey, Dao, and Todorov, 2016 described above it was shown that it is possible to use UKF to control the 26-dof robot in real-time.

Kalman Filter is a method for estimating the state of the dynamic system based on the sequence of the measurements. This method works by combining the previous states of the system with the new measurements to achieve the best evaluation of the current system state. The essential idea of the Kalman Filter lies in using the information from two sources - the previous state of the system and its new measurements - while taking into account their confidential intervals.

The operation of the Kalman Filter can be split into just two essential stages (see Fig. 3.2):

- Prediction: At this stage, the filter uses the system's model to estimate the following states of the system based on the previous state.
- Correction: Succeeding the first stage, the filter combines the predicted state with the new measurements, using the weighted average, to obtain the best estimate of the current state of the system. At this stage, the covariance of the estimation error is also updated.

EKF is an extension of the Kalman Filter for nonlinear systems. Instead of utilizing the system's lineal models and measurements, just as in the basic Kalman Filter, EKF linearizes these models around an estimate of the current mean and covariance, using Taylor series expansions. This approach allows the use of the linear Kalman filter at each step. However, EKF can be unstable and quickly diverge for complex systems owing to its linearization. Since biomechanical models are expected to be highly non-linear, the use of EKF will not help solve the state estimation problem.

UKF is an alternative method for evaluating nonlinear systems that avoid analytical linearization of systems. Instead, it uses statistical linearization approximating state distribution by a Gaussian random variable. This allows you to avoid linearization problems and more accurately model nonlinear systems, capturing the posterior mean and covariance accurately for any nonlinearity (up to the third order of Taylor series expansion). Therefore, UKF is an extension of the Kalman Filter that allows us to evaluate the operation systems and is an excellent option for evaluating states in the developed framework.

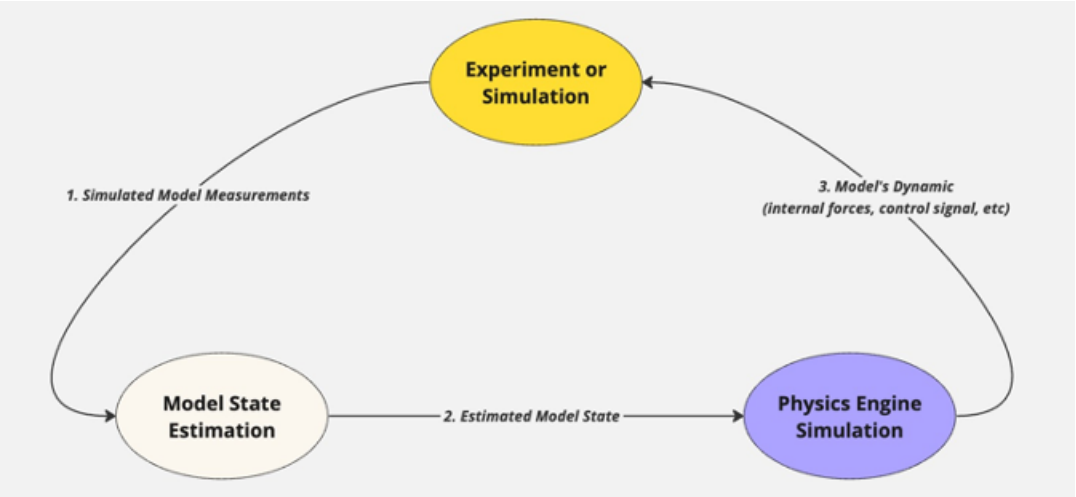

FIGURE 3.3: Proposed dynamic transformations of biomechanical models framework structure.

3.4 Summary

The core purpose of the current project is to design a cloud-based tool to calculate dynamic transformations based on streamed biomechanical signals in real time with the possibility of collecting the received data. As a result, the framework will consist of three parts (see Fig. 3.3):

1. The Client wraps a real-world experiment or simulation and is responsible for collecting and sending measurements to the server side.

2. Model State Estimation module - the part of the server responsible for short-term predictions of the simulation state in case the latency of simulation frequency data exchange is exceeded.

3. Physics Engine module - the part of the server responsible for calculating practical dynamics transformations using the API of the physics engine and storing the historical data.

In the process of developing the pipeline for data streaming, the necessity of subsequent data processing in real-time should be taken into account. This means that there is a need to minimize network latency when streaming so that they have the least possible impact on the overall experience. A possible way to achieve this would be a minimization of the size of the sent data, as well as using protocols with a larger payload but with the possible loss of packets. The only mandatory requirement for the data exchange is to guarantee the integrity of the package. Also, the developed pipeline must provide sufficient functionality to describe the model under study (model's metadata).

When organizing a data warehouse, it is important to take into account the specificity of the data being stored. Biomedical signals recorded in laboratories are taken from complex biomechanical structures that represent high-volume data. At the same time, data is often collected at high rates, and the volumes of collected data are also quite large. It is also necessary to remember about the anonymization of data.

Chapter 4

Experiments

The process of calculating the dynamics of the model with the use of a physics engine is an engineering problem - the correct decision on technologies, parallelization of calculations, ensuring the integrity of shared data, etc. Measuring the time spent on calculations is crucial to understanding the solution's capabilities fully. However, the primary module of the framework that needs to be tested is the state estimation module based on UKF. In experiments, it is needed to test how accurately it can predict the state of the model to compensate for network latencies.

In addition to the standard examination of the operability of the model in the selected simulation, it is necessary to check the estimator on the simple model, the complexity of which can be scaled up in order to make a comparison of the results with each other. The simplest option to conduct a series of experiments such as this would be to use a mathematical pendulum with a different number of links (see Fig. 4.1). For this purpose, a pendulum model was created consisting of a rod with a radius of 0.05 meters, a length of 1 meter, and a mass of 1 kilogram connected by frictionless fasteners.

Furthermore, during the experiment, it is necessary to test several scenarios of working with the framework. Altogether, there are three scenarios for interaction with the model:

1. Constant velocity models that move along the predictable trajectories (their speed changes slowly or does not change at all) (passive walking system, for instance).

2. Models that utilize a control signal to perform specified actions, which we can measure. Moreover, both options are suitable when the framework generates such a signal or comes from the research object.

3. Models that are subject to external forces that we cannot model or measure (capture data from actual human limbs, for example).

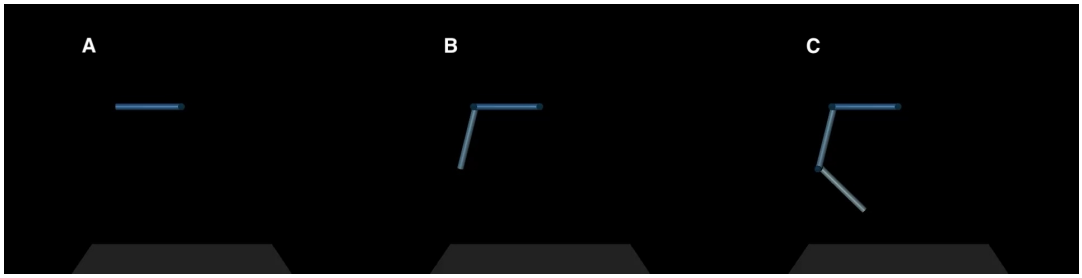

FIGURE 4.1: Pendulum models built in MuJoCo. A - 1-dof pendulum, B - 2-dof pendulum, C - 3-dof pendulum.

The mentioned scenarios need to be separated due to the necessity of being simulated differently using the Kalman Filters. First of all, this affects the parameters that are included in the Kalman Filters state, the generation of noise for process noise (a measure of the sensitivity of the internal state to new measurements), and the choice of other hyperparameters.

As an algorithm for calculating sigma points, the classic version presented by Julier, 2002; Wan and Van Der Merwe, 2000 is used. It has α, β, κ parameters as a hyperparameters:

- α is responsible for the spread of the sigma points around the mean;
- κ secondary scaling parameter responsible for the spread of the sigma points around the mean;
- β is responsible for incorporating prior knowledge of the input distribution.

The values for these hyperparameters are taken from the recommended ones by Julier, 2002; Wan and Van Der Merwe, 2000:

- α - small positive value, $\alpha = 0.001$;
- $\kappa = 3 - n$, where n is the dimensionality of the state or $\kappa = 0$;
- $\beta = 2$, this value is optimal for the Gaussian distribution and will be changed if necessary.

An essential part of setting up Kalman Filters is the initial values of state estimate vector x and covariance estimate matrix P. Since the experiment environment is completely controlled, we can set the real starting value of the experiment in x and, accordingly, P with low initial uncertainty. Thanks to this, the Kalman Filter will converge much faster, and this will not affect estimation (average of RMSE, for example).

Another vital part of setting up Kalman Filters is constructing the measurement noise matrix R and process noise matrix Q. For the measurement noise matrix R, the situation with state estimate vector x is repeated - the simulation is completely controlled, and the measurements do not contain noise (the sensors are absolutely accurate). Accordingly, the variance of noise measurement will be low. For the process noise matrix Q, a discrete noise model was chosen since the simulation frequency is relatively high (for the pendulum model - 50ms). At the same time, the variance in the noise for Q varies depending on the model and experiment since this is important for capture maneuvers.

The filter performance must evaluate the obtained state estimation results. In addition to the expected evaluating methods based on residuals (RMSE, MAE, etc.), due to the fact that we have a controlled simulation, we can use Normalized Estimated Error Squared (NEES). This method is one of the simplest and, at the same time, robust methods, but it is only suitable for simulating systems in which we know the actual value. Let $\mathbf{x}$ be the true state, $\hat{\mathbf{x}}$ - Kalman Filter state estimate and $\mathbf{P}$ - covariance matrix, then:

$$\tilde{\mathbf{x}} = \mathbf{x} - \hat{\mathbf{x}}$$

$$\epsilon = \tilde{\mathbf{x}}^T \mathbf{P}^{-1} \tilde{\mathbf{x}}$$

The main idea of NEES is based on the fact that the estimation error $\tilde{\mathbf{x}}$ shall be zero in an ideal Kalman Filter:

$$E[[\mathbf{x} - \hat{\mathbf{x}}]] = E[\tilde{\mathbf{x}}] \stackrel{!}{=} 0$$

And the covariance matrix $\mathbf{P}$ shall be equal to the covariance matrix of the estimation error:

$$E[[\mathbf{x} - \hat{\mathbf{x}}][\mathbf{x} - \hat{\mathbf{x}}]^T] = E[\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T] \stackrel{!}{=} \mathbf{P}$$

Thus, given ϵ is chi-squared distributed with n degrees of freedom, we can test the hypothesis:

H_0: The estimation error $\tilde{\mathbf{x}}$ is consistent with the covariance matrix $\mathbf{P}$.

4.1 Pendulum Model

4.1.1 Constant Velocity Pendulum

The first experiment is carried out on models of a pendulum (see Fig. 4.1) on which no external forces act, with the exception of the force of gravity. The simulation runs at a high frequency of 50ms, and the pendulum model does not change its trajectory abruptly, which results in the UKF being able to capture state changes as if the model had constant velocity and zero acceleration. For such a scenario, it is enough for us to have the following state $\mathbf{x}$ and measurement vector $\hat{\mathbf{x}}$:

$$\mathbf{x} = [\Theta_1, \cdots, \Theta_n, \dot{\Theta}_1, \cdots, \dot{\Theta}_n]$$
$$\hat{\mathbf{x}} = [\Theta'_1, \cdots, \Theta'_n, \dot{\Theta}'_1, \cdots, \dot{\Theta}'_n],$$

where Θ_n - angle of the n-th joint, $\dot{\Theta}_n$ - angular velocity of the n-th joint.

The remaining hyperparameters are located in the Table 4.1. The chosen values for covariance estimate matrix $\mathbf{P}$ are quite small since the initial value of the state estimate vector $\mathbf{x}$ corresponds to the real one. The value of the variance of process noise matrix $\mathbf{Q}$ is also quite low since large perturbations are not expected in the simulation. The results of the experiments are depicted in Fig. 4.2, Fig. 4.3, Fig. 4.4 and Table 4.2.

α	β	κ	dt	$\mathbf{P}$	$\mathbf{Q}$ Var	$\mathbf{R}$
0.001	2	$3 - 2n$, where n - is dof	50 ms	$0.2 * \mathbf{I}$	0.1	$0.01 * diag(\mathbf{I})$

TABLE 4.1: Constant Velocity Pendulums Hyperparameters.

DoF	Θ aRMSE	$\dot{\Theta}$ aRMSE	Average NEES	NEES CI
1	2.3912e-5	1.0518e-4	0.9586e-4	[0.0506; 7.3778]
2	2.0877e-5	1.0.9652e-4	2.2982e-4	[0.4844; 11.1433]
3	3.5730e-5	1.5348e-4	7.9482e-4	[1.2373; 14.4494]

TABLE 4.2: Constant Velocity Pendulums Errors.

As we can see, the predicted value completely repeats the actual values of the angles and angular velocities of the pendulums without the influence of the complexity of the model. This is confirmed by the extremely small average RMSE values

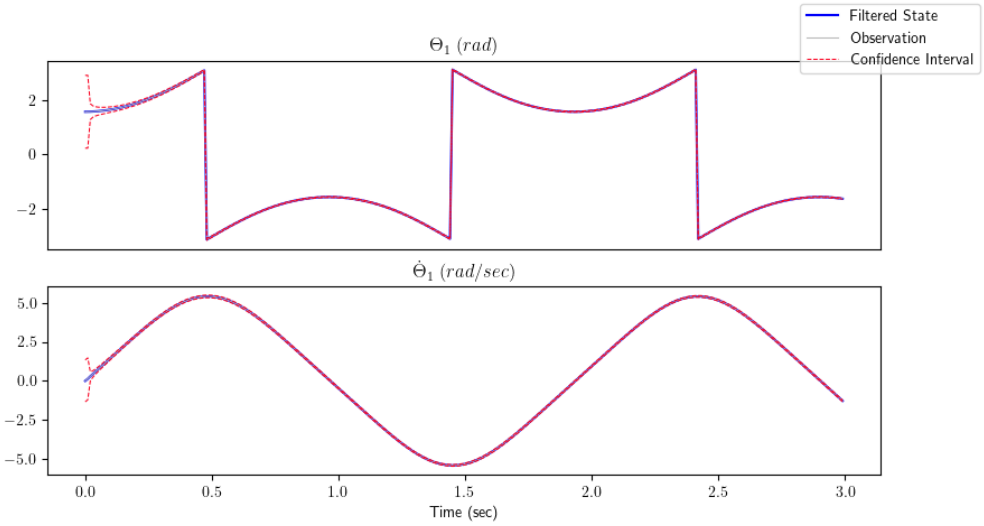

FIGURE 4.2: 1-dof Constant Velocity Pendulum State Estimation.

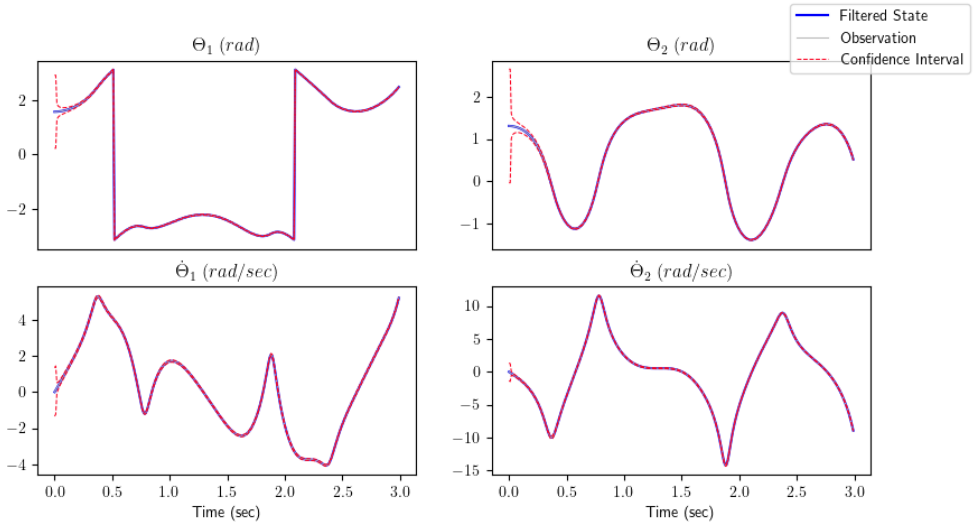

FIGURE 4.3: 2-dof Constant Velocity Pendulum State Estimation.

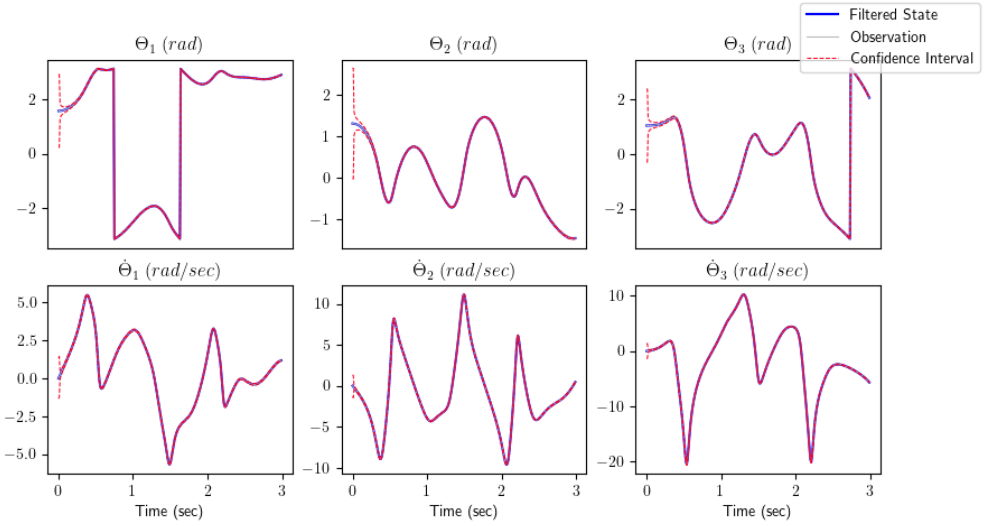

FIGURE 4.4: 3-dof Constant Velocity Pendulum State Estimation.

for angles and angular velocities. The NEES values are below the lower confidence interval limit, which indicates that the variance values for process noise **Q** are greater than necessary, given that measurement noise **R** is modeled correctly. For simulations with such stability, this seems perfectly normal. As a result, we can say that for constant velocity simulations, UKF can provide sufficiently accurate estimations for subsequent calculations of custom dynamics data.

4.1.2 Controlled Pendulum

Afterward, I would like to test the pendulum scenario for which a control signal is generated on the "server" side. Such a case requires a completely different approach to forming the UKF state. In this scenario, it is necessary to capture sudden changes in trajectories. To achieve this, we need to estimate the angular acceleration of the model additionally. In the given scenario, the pendulum will change its position every second to a symmetrical position under the influence of a control signal. For such a case the state $\mathbf{x}$ UKF and the measurement vector $\hat{\mathbf{x}}$ will have the following form:

$$\mathbf{x} = [\Theta_1, \cdots, \Theta_n, \dot{\Theta}_1, \cdots, \dot{\Theta}_n, \ddot{\Theta}_1, \cdots, \ddot{\Theta}_n]$$
$$\hat{\mathbf{x}} = [\Theta'_1, \cdots, \Theta'_n, \dot{\Theta}'_1, \cdots, \dot{\Theta}'_n, \ddot{\Theta}'_1, \cdots, \ddot{\Theta}'_n],$$

where Θ_n - angle of the n-th joint, $\dot{\Theta}_n$ - angular velocity of the n-th joint, $\ddot{\Theta}_n$ - angular acceleration of the n-th joint.

α	β	κ	dt	**P**	**Q** Var	**R**
0.001	2	$3 - 3n$, where n - is dof	50 ms	$0.2 * \mathbf{I}$	0.2	$0.01 * diag(\mathbf{I})$

TABLE 4.3: Controlled Pendulums Hyperparameters.

The hyperparameters for experiments are indicated in the Table 4.3. In the following case, hyperparameters are slightly modified to better comply with the simulation conditions. The value of the κ for the algorithm for calculating sigma points has changed since for each joint now we have three parameters in the state - Θ_n, $\dot{\Theta}_n$ and $\ddot{\Theta}_n$, and not two parameters as for the previous experiment. The value for the variance of process noise matrix **Q** is also changed - it became larger. This decision was made due to the fact that the simulation would have more perturbations compared to the previous experiment. Additionally, the control signal is calculated based on the UKF state, which does not fully correspond to the real state, which can cause deviations between the research object and its estimated state. The results of the experiments are depicted on the Fig. 4.5, Fig. 4.6, Fig. 4.7 and Table 4.4.

DoF	Θ aRMSE	$\dot{\Theta}$ aRMSE	$\ddot{\Theta}$ aRMSE	Average NEES	NEES CI
1	6.3058e-5	6.7594e-5	1.8365e-3	0.8221e-3	[0.982e-03; 9.348]
2	1.4820e-5	6.0486e-5	0.8388e-3	1.1257e-3	[1.2373; 14.4494]
3	3.4770e-5	8.2194e-5	1.0114e-3	1.8903e-3	[2.7004; 19.0228]

TABLE 4.4: Controlled Pendulums Errors.

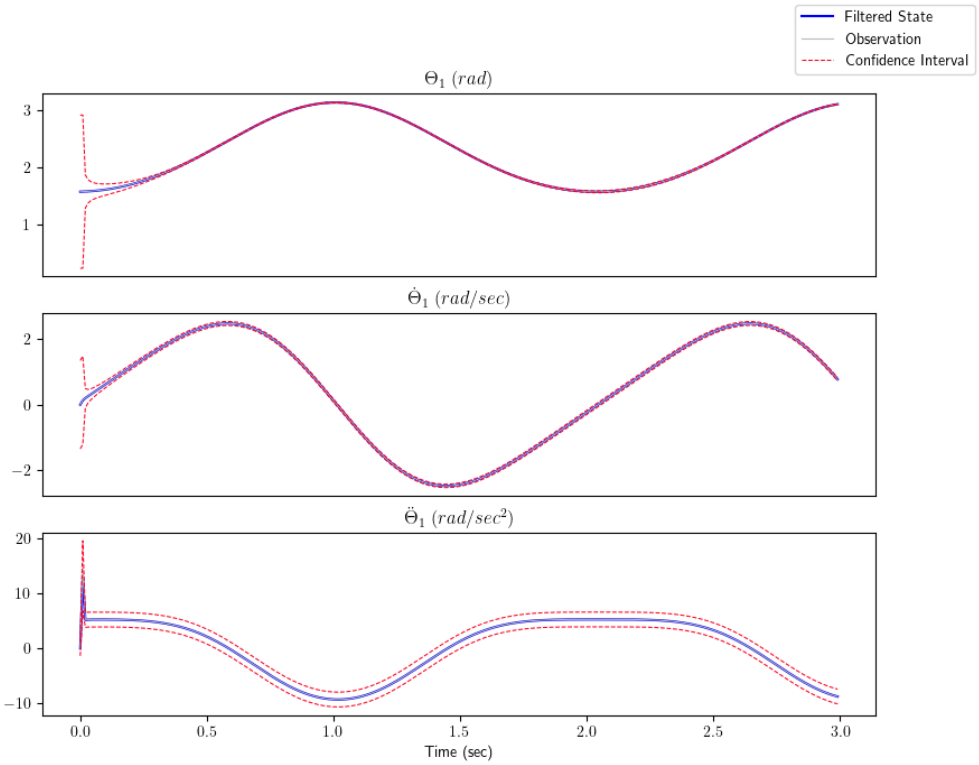

FIGURE 4.5: 1-dof Controlled Pendulum State Estimation.

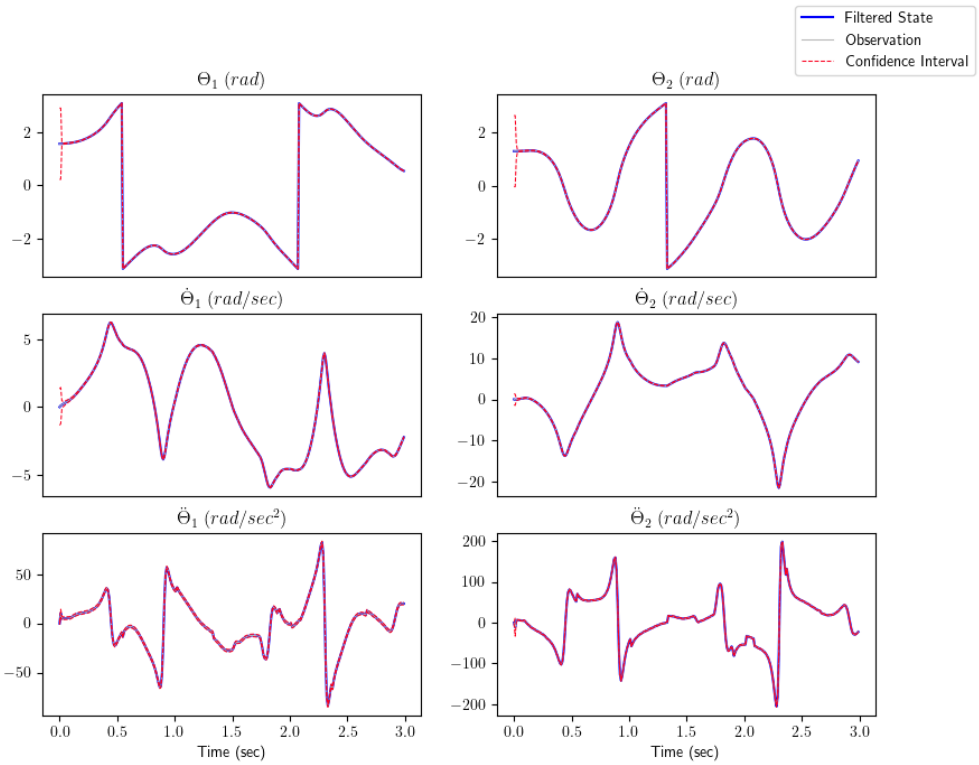

FIGURE 4.6: 2-dof Controlled Pendulum State Estimation.

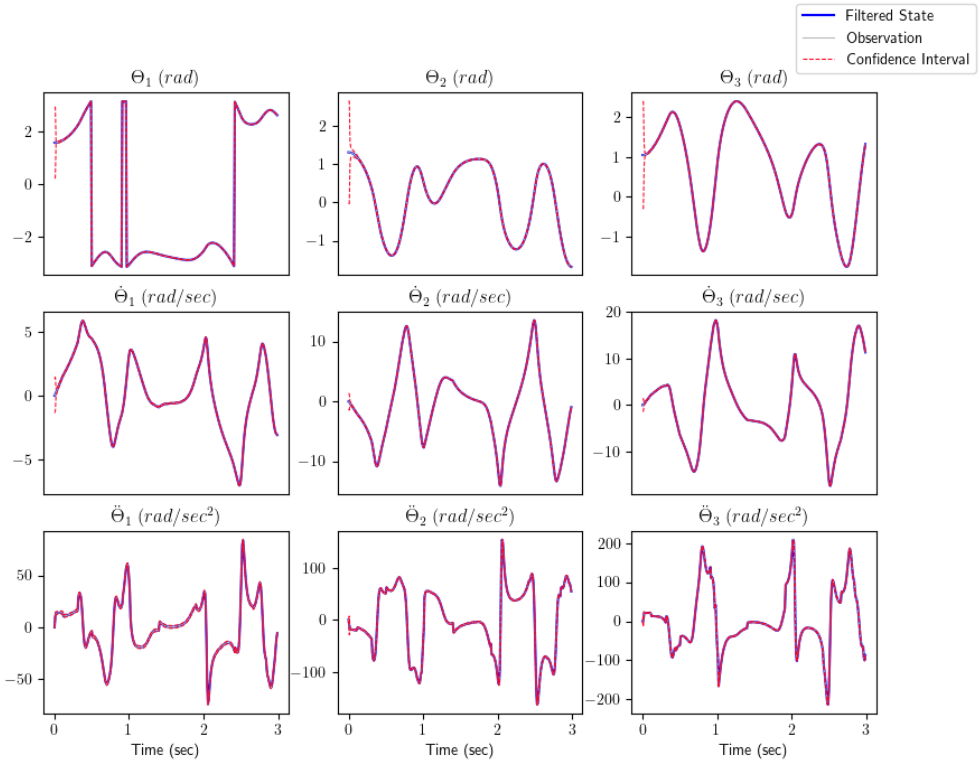

FIGURE 4.7: 3-dof Controlled Pendulum State Estimation.

The results of this experiment turned to be similar to the results of the previous experiment:

- State Estimation repeats the actual values with high accuracy.
- Average Θ and $\dot{\Theta}$ RMSE are small.
- The NEES values are below the lower limit of the confidence interval (process noise overestimated). However, they are already closer to the lower limit compared to the previous experiment.

As for the $\ddot{\Theta}$, the confidence interval of $\ddot{\Theta}$ if much wider, average $\ddot{\Theta}$ RMSE is several orders of magnitude higher than the deviations Θ and $\dot{\Theta}$, however this is still enough for calculations with the use of the Physics Engine. In general, we can conclude that the suggested approach with the use of the UKF is suitable for the given scenario.

4.1.3 Externally Controlled Pendulum

The following experiment has a similar logic to the controlled pendulum, except that during the prediction process, there is no access to the control signal for the UKF. As a result, the UKF is not able to simulate the force applied to the pendulum and is only able to adjust the state of the filter after receiving measurements. In order to compensate for this, we can use the Physics Engine feature, calculate inverse dynamics, and restore the previous force that acted on the object. Taking into account that the object will not rapidly change its trajectory, we can have a reasonably accurate

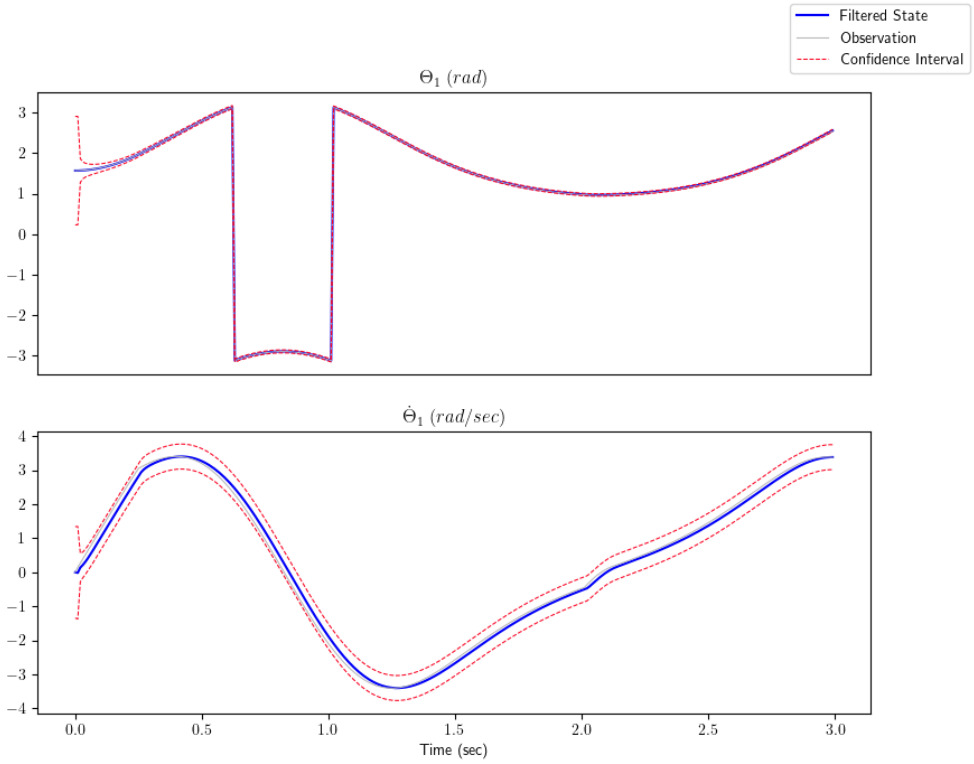

FIGURE 4.8: 1-dof Externally Controlled Pendulum State Estimation.

estimation. To model such a process, we need the same state as for constant velocity pendulum pendulum:

$$\mathbf{x} = [\Theta_1, \cdots, \Theta_n, \dot{\Theta}_1, \cdots, \dot{\Theta}_n]$$
$$\hat{\mathbf{x}} = [\Theta'_1, \cdots, \Theta'_n, \dot{\Theta}'_1, \cdots, \dot{\Theta}'_n]$$

α	β	κ	dt	$\mathbf{P}$	$\mathbf{Q}$ Var	$\mathbf{R}$
0.001	2	$3 - 2n$, where n - is dof	50 ms	$0.2 * \mathbf{I}$	50	$0.01 * diag(\mathbf{I})$

TABLE 4.5: Externally Controlled Pendulums Hyperparameters.

In comparison with the previous experiments, there will be significantly more unknowns in the estimation, which is why we cannot use any of the hyperparameters from the previous experiments. The final hyperparameters after the selection are in the Table 4.5. To successfully model this simulation, it was necessary to increase the variance of process noise $\mathbf{Q}$ up to a significantly large value. The final results are displayed in the Fig. 4.8, Fig. 4.9, Fig. 4.10 and Table 4.6.

DoF	Θ aRMSE	$\dot{\Theta}$ aRMSE	Average NEES	NEES CI
1	6.3386e-3	0.113	2.0765	[0.0506; 7.3778]
2	55.2721e-3	0.4826	11750.1188	[0.4844; 11.1433]
3	71.3011e-3	0.5848	3488.8626	[1.2373; 14.4494]

TABLE 4.6: Externally Controlled Pendulums Errors.

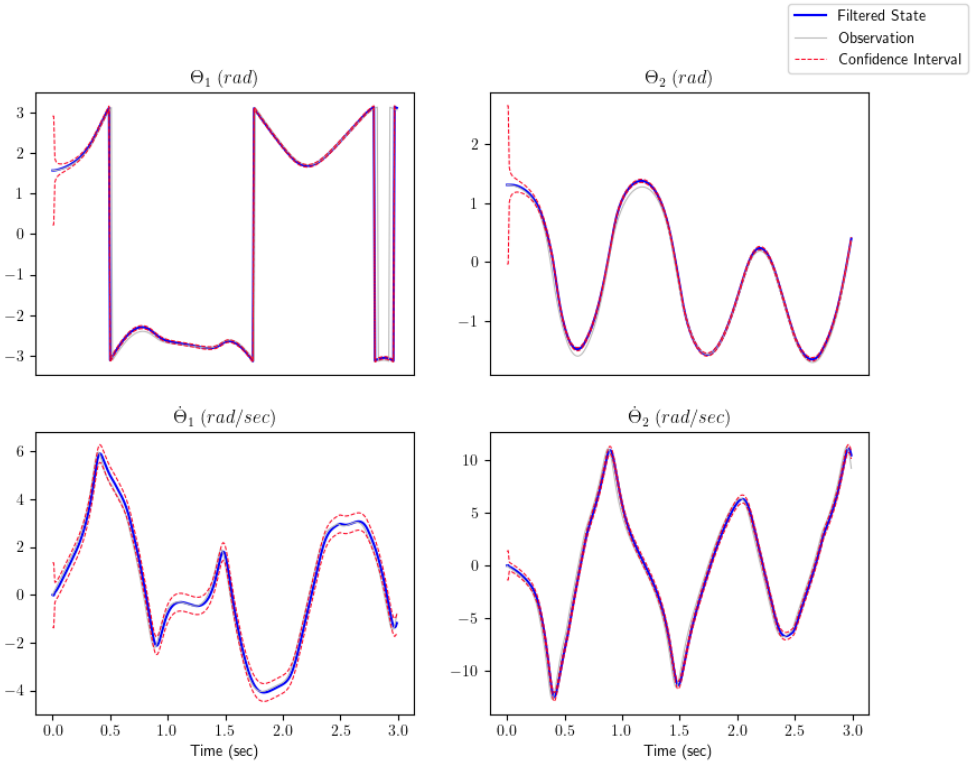

FIGURE 4.9: 2-dof Externally Controlled Pendulum State Estimation.

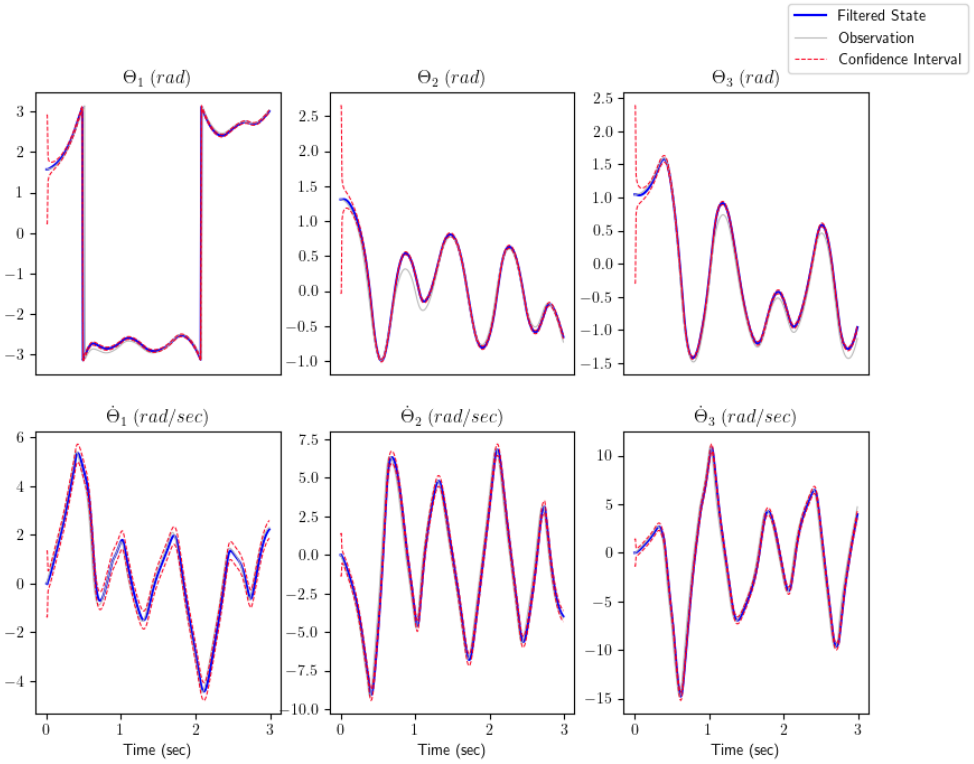

FIGURE 4.10: 3-dof Externally Controlled Pendulum State Estimation.

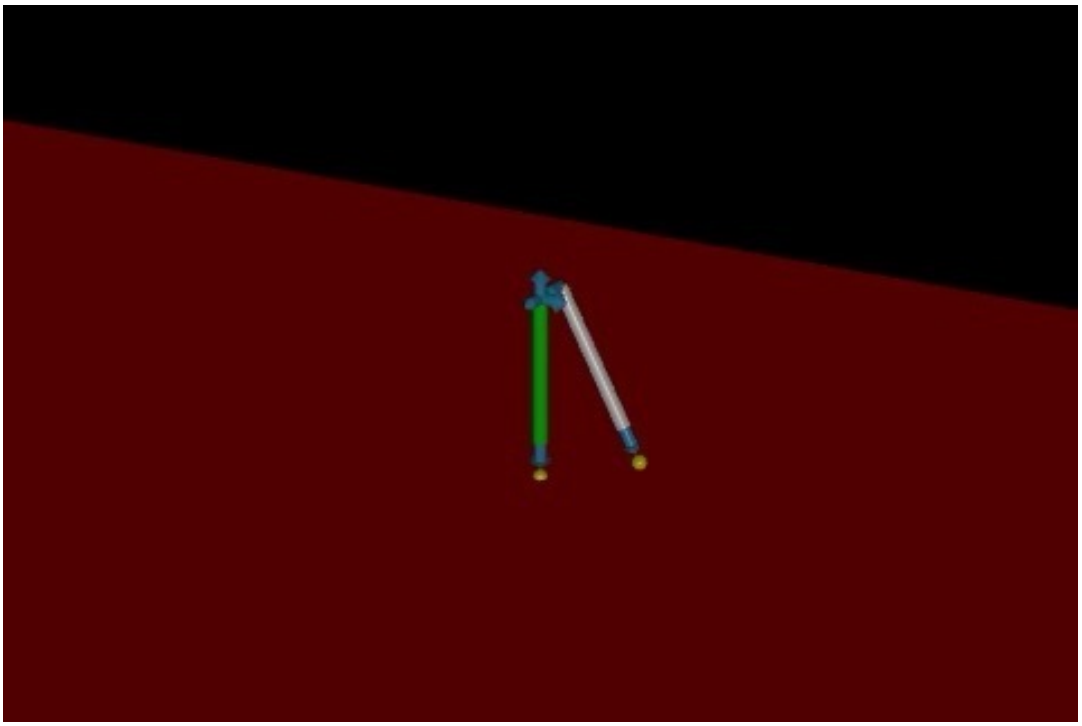

FIGURE 4.11: Biped model built in MuJoCo.

As can be seen, the obtained results do not so perfectly replicate the actual values, even though the results obtained are still quite accurate. Average Θ RMSE does not exceed 4.09°, which, as I consider, is a fairly good result. We are also now able to see an explicit dependency between errors and the amount of dof - the larger the dof, the higher the RMSE. Furthermore, with an extreme value of the variance of process noise $\mathbf{Q}$, NEES for the 2-dof and 3-dof is higher than the upper limit of the confidence interval, which means that this value is not yet sufficient.

4.2 Biped Model

The next phase of the experiments on UKF is to move to models with a large number of steps of freedom and with contact with the surface. As an example of such a model, I chose a biped (See Fig. 4.11), which consists of six joints (two hinge joints and four slide joints) and two rods with a radius of 0.05 meters, a length of 1 meter and a mass of 1 kilogram. The chosen model is already much more complicated since it contains a larger number of joints that have different types (different measurement units), and there is contact with the ground, so for satisfactory results, it was necessary to reduce the frequency of simulations to 5 ms. The model is also not capable of passive movement (only falling), so two scenarios will be tested: controlled biped and externally controlled biped.

4.2.1 Controlled Biped

The following experiment is identical to the experiment with the controlled pendulum: biped simulation controlled by a signal calculated from the data from the Kalman Filter state. The state, in this case, is much more complex not only for angular units and derivatives but also for values to describe slide joints (measured in

meters). Generally, the state $\mathbf{x}$ UKF and measurement vector $\hat{\mathbf{x}}$ will look the following:

$$\mathbf{x} = [x_1, \cdots, x_n, \dot{x}_1, \cdots, \dot{x}_n, \ddot{x}_1, \cdots, \ddot{x}_n]$$
$$\hat{\mathbf{x}} = [x'_1, \cdots, x'_n, \dot{x}'_1, \cdots, \dot{x}'_n, \ddot{x}'_1, \cdots, \ddot{x}'_n],$$

where x_1 - x axis position, x_2 - z axis position, x_3 - value of the left leg hinge, x_4 - left leg shock absorber, x_5 - value of the right leg hinge, x_6 - right leg shock absorber. The hyperparameters of the experiment are located in the Table 4.7.

	Controlled Biped	Externally Controlled Biped
α	0.001	0.001
β	2	2
κ	0	0
dt	5 ms	5 ms
$\mathbf{P}$	$0.2 * \mathbf{I}$	$0.2 * \mathbf{I}$
$\mathbf{Q}$ Var	5	50
$\mathbf{R}$	$0.01 * diag(\mathbf{I})$	$0.01 * diag(\mathbf{I})$

TABLE 4.7: Controlled and Externally Controlled Bipeds Hyperparameters.

In accordance with the results of the experiment (See Fig. 4.12 and Table 4.8), UKF can successfully perform state estimation for much more complex models with contact. The same situation is repeated as for the controlled pendulum - variance of process noise $\mathbf{Q}$ is overestimated (NEES is below the confidence interval). The main challenge is the necessity of increasing the simulation frequency to 5 ms in order to be able to generate the control signal using MuJoCo (numerical integration limitation).

№	RMSE of x	RMSE of $\dot{x}$	RMSE of $\ddot{x}$	Average NEES	NEES CI
1	14.0307e-6	4.6299e-5	2.054e-3	12.7856e-3	[8.2308; 31.5264]
2	3.7699e-6	2.3872e-5	7.6856e-3		
3	8.3939e-6	4.0312e-5	7.1893e-3		
4	0.6027e-6	10.9725e-5	96.4136e-3		
5	2.5343e-6	2.8387e-5	12.7e-3		
6	0.1032e-6	0.3612e-5	6.8851e-3		

TABLE 4.8: Controlled Biped Errors.

4.2.2 Externally Controlled Biped

An experiment with biped, which is controlled by an external control signal, replicates the setup of an experiment with externally controlled only with an increased simulation frequency (hyperparameters for the experiment in Table 4.7) and with a state of the following form:

$$\mathbf{x} = [x_1, \cdots, x_n, \dot{x}_1, \cdots, \dot{x}_n]$$
$$\hat{\mathbf{x}} = [x'_1, \cdots, x'_n, \dot{x}'_1, \cdots, \dot{x}'_n]$$

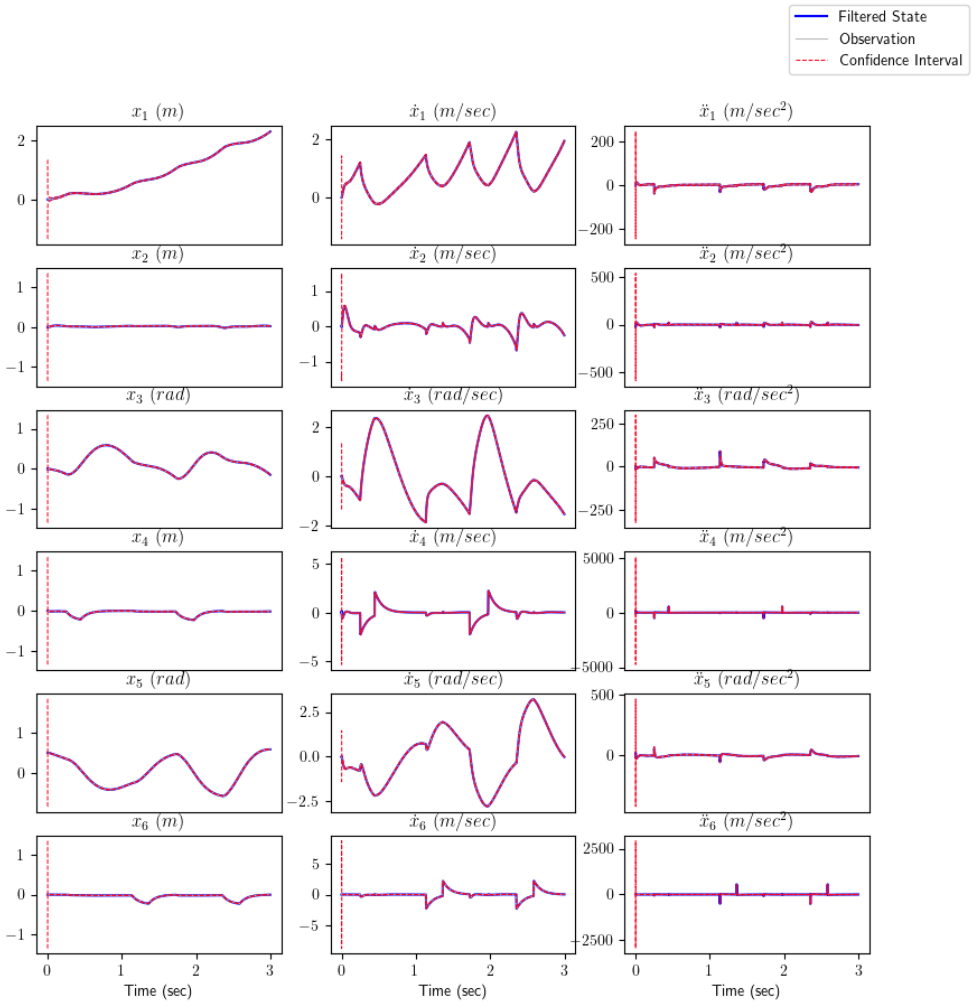

FIGURE 4.12: Controlled Biped State Estimation.

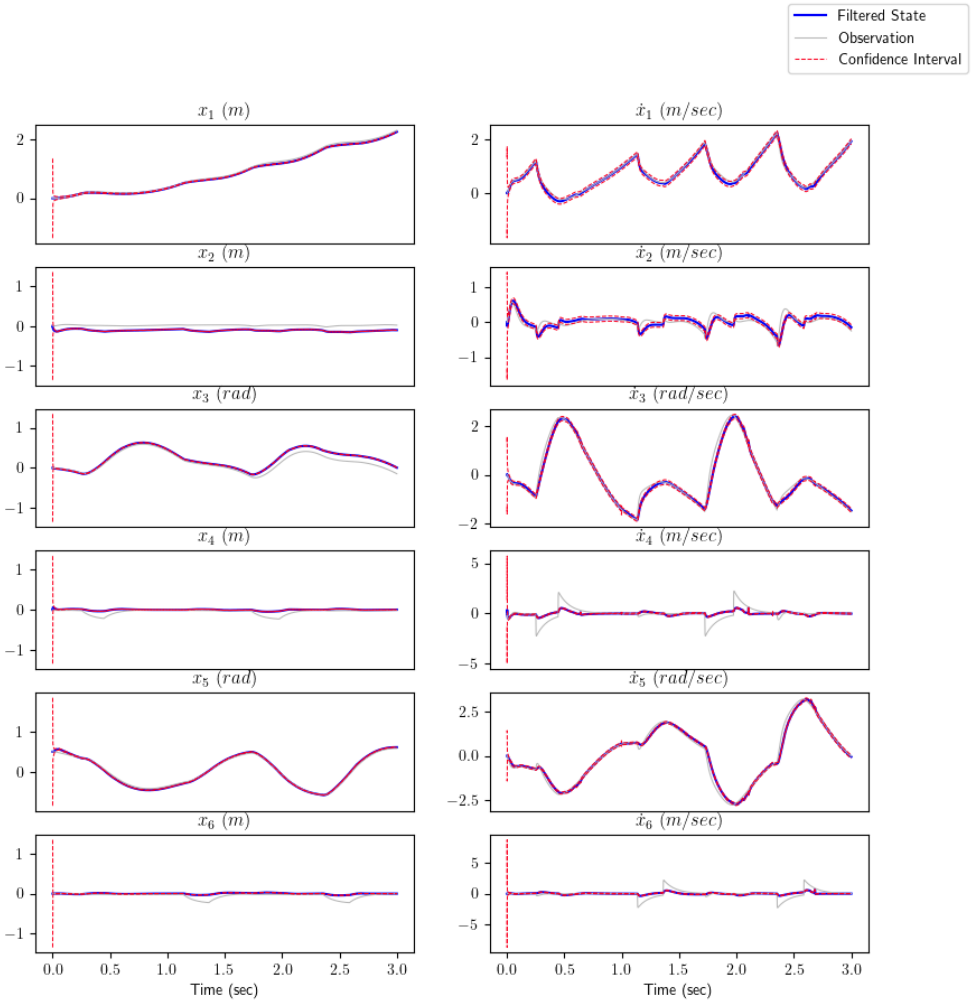

FIGURE 4.13: Externally Controlled Biped State Estimation.

The obtained results (See Fig. 4.13 and Table 4.9) are pretty accurate. The UKF is good at predicting the value for the hinge joint and the x-axis position, but it is systematically wrong about the remaining joints responsible for changes in the z-axis. Visually, the simulation based on predicted values looks smooth, periodically deviating from the true simulation and gradually returning back to it.

№	RMSE of x	RMSE of $\dot{x}$	Average NEES	NEES CI
1	3.9037e-2	4.4376e-2	859631.6084	[4.4038; 23.3367]
2	12.6587e-2	10.1083e-2		
3	8.033e-2	11.1617e-2		
4	4.5617e-2	30.0764e-2		
5	2.3774e-2	13.3222e-2		
6	4.7128e-2	30.3672e-2		

TABLE 4.9: Externally Controlled Biped Errors.

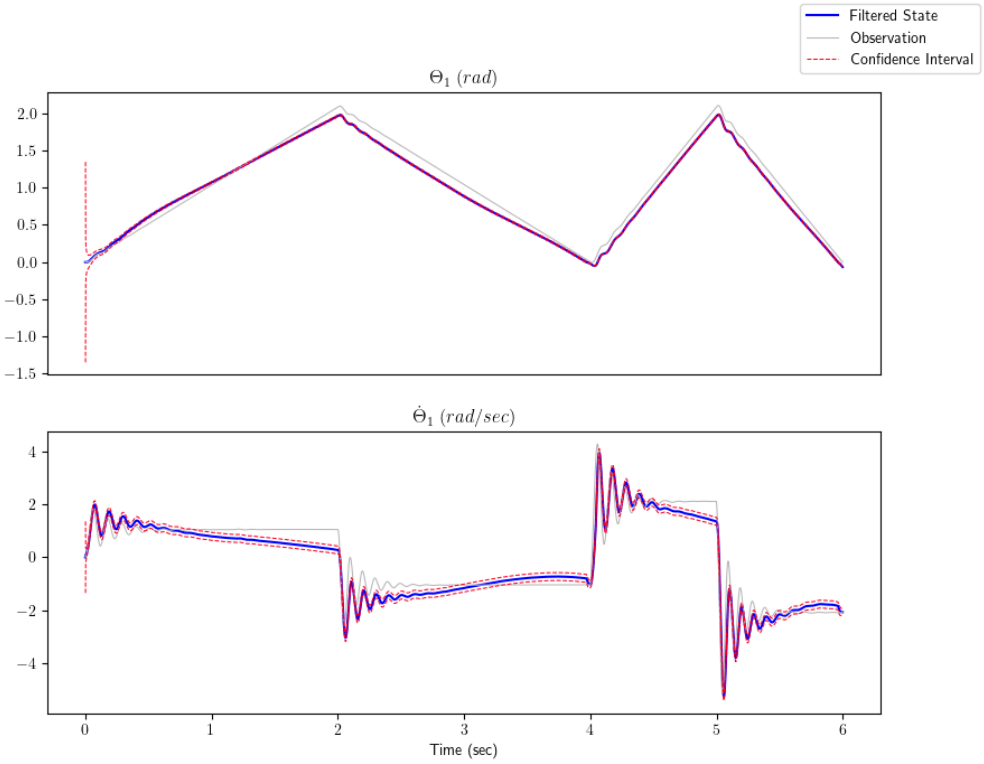

FIGURE 4.14: Externally Controlled MyoElbow State Estimation.

4.3 Biomechanical Models

The final part of the experiments with state estimate with UKF is its testing on biomechanical models from MyoSuite. The experiment was carried out on two models: MyoElbow and MyoHand (See Fig. 3.1). The MyoElbow model as the simplest model with 1-dof, and the MyoHand model as a highly complex model with 23-dof. A real-world scenario, when the biomechanical model will make some movements and Kalman Filter will not have access to the control signal (capturing data from the subject limb), was tested. For the MyoElbow model, this is a gradual raising and lowering of the forearm; for the MyoHand model, this is alternate bending of the fingers. The hyperparameters were reused from the experiment with externally controlled biped (Table 4.8) only with an even higher simulation frequency - 2 ms. For both models, the state $\mathbf{x}$ and measurement vector $\hat{\mathbf{x}}$ look like this (description of each joint is omitted due to the complexity of the MyoHand):

$$\mathbf{x} = [\Theta_1, \cdots, \Theta_n, \dot{\Theta}_1, \cdots, \dot{\Theta}_]$$
$$\hat{\mathbf{x}} = [\Theta'_1, \cdots, \Theta'_n, \dot{\Theta}'_1, \cdots, \dot{\Theta}'_n]$$

RMSE of x	RMSE of $\dot{x}$	Average NEES	NEES CI
9.4534e-2	2.9799e-1	1160.0789	[0.0506; 27.3778]

TABLE 4.10: Externally Controlled MyoElbow Errors.

The elbow movements were captured quite well (See Fig. 4.14 and Table 4.10), while for the hand model, it was not possible to achieve acceptable results. During

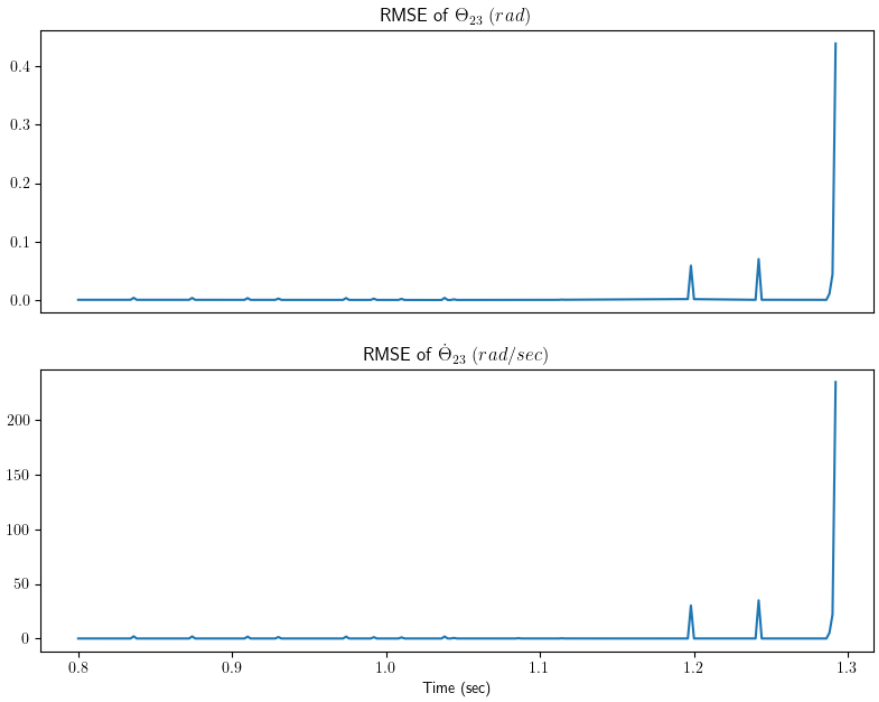

FIGURE 4.15: Externally Controlled MyoHand RMSE of 23-rd joint.

the MyoHand simulation, UKF accumulated an error and began to diverge very quickly (See Fig. 4.15). In general, we can see that estimating a biomechanical model state using UKF is possible, but only for fairly simple body parts, such as the elbow. For models comparable in complexity to MyoHand, you need to look for another method.

4.4 State Estimation Duration

The last aspect that needs to be assessed is the overhead that UKF creates as a state estimator. The UKF contains several resource-intensive operations:

- In the state prediction step, the UKF calculates $2n + 1$ sigma points, where n - is the number of state dimensions to approximate the state. That is, at each step, it is necessary to solve $2n + 1$ dynamics equations.
- In the state update step, UKF calculates the inverse of the weighted variance of the measurement matrix, which can be a fairly lengthy operation for high-dimensional problems.

If we cannot significantly influence the inverse of a matrix, then the process of calculating sigma points can be parallelized. There is no need for sequential calculations of sigma points since these are completely independent processes. Thus, the calculation of each point can be run on a separate thread, reducing the calculation time by several times.

As we can see in the graph Fig. 4.16, increasing the number of processes allows us to reduce the amount of network time by one step. We can also see that adding

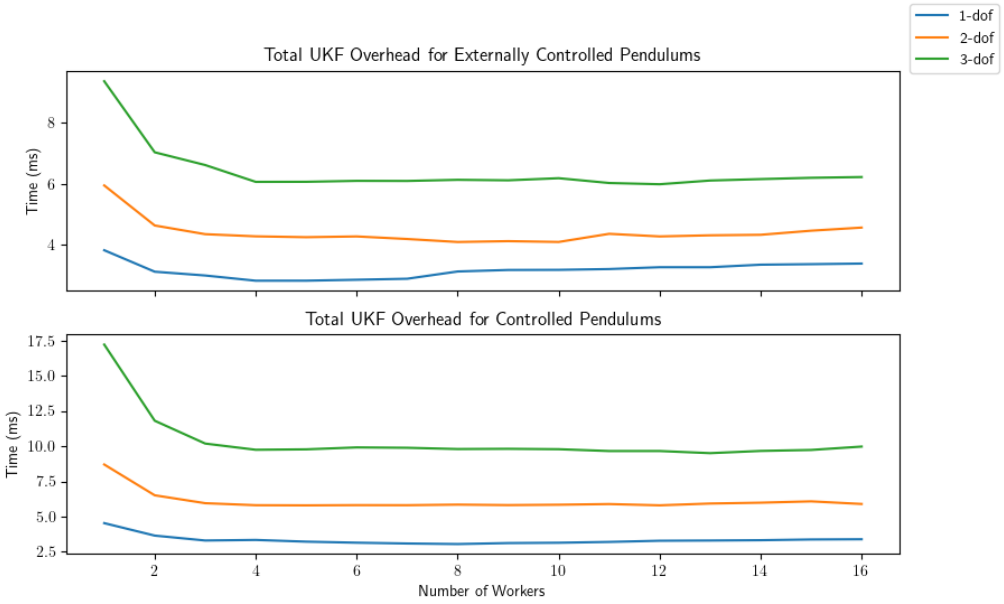

FIGURE 4.16: UKF overhead for Pendulums Simulations.

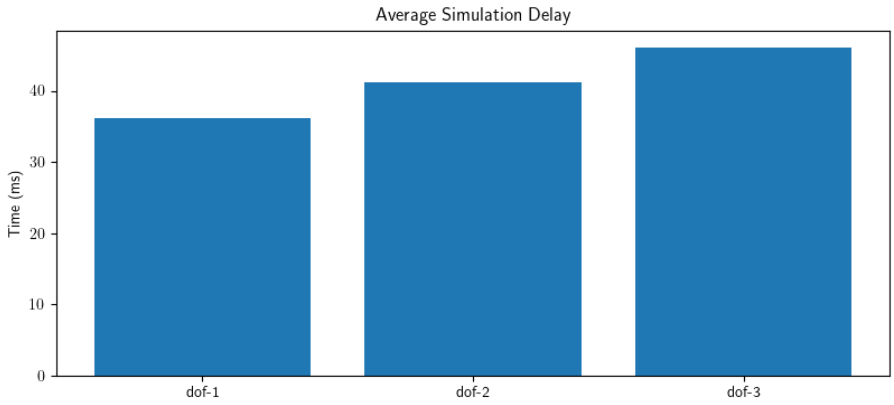

FIGURE 4.17: Average delays of Constant Velocity Pendulums.

angular velocity $\ddot{\Theta}$ to the state in Controlled Pendulums significantly increases UKF overhead. Despite this, in the worst case, the 3-dof Controlled Pendulum, overhead is 10 ms, which gives another 40 ms for useful calculations.

Fig. 4.17 shows the average total delays for simulating constant-velocity pendulums during network interaction with the cloud. We can see that the average network latencies were about 30-35ms, and the rest is overhead from the framework. All pendulums, on average, have a total delay of less than the simulation frequency.

Chapter 5

Conclusions

5.1 Discussion

In the present work, the capabilities of the Physics Engine and UKF combination were tested in order to solve the problem of just-in-time dynamics computations in the cloud. The developed framework was tested in different scenarios and on models of varying complexity. The proposed method performed well on models of low and medium complexity in terms of accuracy. For models of very high complexity, such as the MyoHand model, it is necessary to look for other ways to compensate for network lags if it is needed to achieve real-time.

Taking into consideration the modern capabilities of networks with stable latencies of no more than 40 - 50 ms, and in some cases up to 10 ms, it is possible to simulate a wide range of scenarios in real time using the proposed framework. UKF has compiled well with the cloud's ability to have access to the multi-core CPU, which can help significantly speed up not only state estimation but also potentially custom dynamics computations. A limitation of the proposed method is the inability to achieve real-time in tasks that require a high simulation frequency (5 ms for biped or 2 ms for biomechanical models). However, that is rather a challenge not for the proposed method but for the environment that includes network interaction. In general, achieving real-time dynamics transformations highly depends on the complexity of the model and the complexity of the problem, with no universal method. Cloud computing will significantly expand the existing set of options.

5.2 Future Work

The primary direction of possible future research is related to the search for methods of lag compensation. Such could be applying improvements to Kalman Filters or searching for entirely different methods to produce short-term state estimation since UKF is insufficient for complex biomechanical models. Among the possible enhancements for Kalman Filters, it is necessary to check the influence of signal smoothing, as well as adaptive process and measurement noises on state estimation. At the same time, the results are unlikely to help simulate models similar to MyoHand but may improve accuracy for other models.

Regardless of the success of state estimation, the community needs a standardized solution for dynamics and kinematics problems in the cloud. The experiments demonstrated a vast number of use cases that need to be covered in order to get a production-ready solution. Additionally, special attention calls for the need to calibrate hyperparameters for each task and model. By collecting a more significant number of models and tasks, it is possible to formulate the final requirements for such a platform.

Bibliography

Aimonen, P. (2011). *Basic concept of Kalman filtering*. https://commons.wikimedia.org/wiki/File:Basic_concept_of_Kalman_filtering.svg.

Bernier, Y. W. (2001). "Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization". In: GDC. URL: https://gamedevs.org/uploads/latency-compensation-in-client-server-protocols.pdf.

Bettner, P. and M. Terrano (2001). "1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond". In: GDC. URL: https://zoo.cs.yale.edu/classes/cs538/readings/papers/terrano_1500arch.pdf.

Delp, S. L. et al. (2007). "OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement". In: *IEEE Transactions on Biomedical Engineering* 54.11, pp. 1940–1950. DOI: 10.1109/tbme.2007.901024.

Erez, T. and E. Todorov (2012). "Trajectory optimization for domains with contacts using inverse dynamics". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4914–4919. DOI: 10.1109/iros.2012.6386181.

Featherstone, R. (2008). *Rigid Body Dynamics Algorithms*. Springer eBooks. DOI: 10.1007/978-1-4899-7560-7.

"IEEE Standard for Distributed Interactive Simulation (DIS) – Communication Services and Profiles - Redline" (1996). In: *IEEE Std 1278.2-2015 (Revision of IEEE Std 1278.2-1995) - Redline*, pp. 1–90.

Julier, S. J. (2002). "The scaled unscented transformation". In: *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*. Vol. 6, pp. 4555–4559. DOI: 10.1109/ACC.2002.1025369.

Lowrey, K., J. Dao, and E. Todorov (2016). "Real-time state estimation with whole-body multi-contact dynamics: A modified UKF approach". In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pp. 1225–1232. DOI: 10.1109/humanoids.2016.7803426.

Manukian, M., S. Bahdasariants, and S. Yakovenko (2023). "Artificial physics engine for real-time inverse dynamics of arm and hand movement". In: *PLOS ONE* 18.12. DOI: 10.1371/journal.pone.0295750.

Mauve, M. et al. (2004). "Local-Lag and Timewarp: Providing Consistency for Replicated Continuous Applications". In: *IEEE Transactions on Multimedia* 6.1, pp. 47–57. DOI: 10.1109/tmm.2003.819751.

Pizzolato, C. et al. (2016). "Real-time inverse kinematics and inverse dynamics for lower limb applications using OpenSim". In: *Computer Methods in Biomechanics and Biomedical Engineering* 20.4, pp. 436–445. DOI: 10.1080/10255842.2016.1240789.

Saha, O. and P. Dasgupta (2018). "A Comprehensive Survey of Recent Trends in Cloud Robotics Architectures and Applications". In: *Robotics* 7.3, p. 47. DOI: 10.3390/robotics7030047.

Savery, C. and T. C. N. Graham (2012). "Timelines: simplifying the programming of lag compensation for the next generation of networked games". In: *Multimedia Systems* 19.3, pp. 271–287. DOI: 10.1007/s00530-012-0271-3.

Seth, A. et al. (2018). "OpenSim: Simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement". In: *PLOS Computational Biology* 14.7. DOI: 10.1371/journal.pcbi.1006223.

Sharkey, P. M., M. D. Ryan, and D. J. Roberts (2002). "A local perception filter for distributed virtual environments". In: *Proceedings. IEEE 1998 Virtual Reality Annual International Symposium (Cat. No.98CB36180)*, pp. 242–249. DOI: 10.1109/vrais.1998.658502.

Todorov, E. (2014). "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6054–6061. DOI: 10.1109/icra.2014.6907751.

Todorov, E., T. Erez, and Y. Tassa (2012). "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 5026–5033. DOI: 10.1109/iros.2012.6386109.

Vittorio, C. et al. (2022). "MyoSuite – A contact-rich simulation suite for musculoskeletal motor control". In: DOI: 10.48550/ARXIV.2205.13600.

Wan, E. A. and R. Van Der Merwe (2000). "The unscented Kalman filter for nonlinear estimation". In: *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pp. 153–158. DOI: 10.1109/ASSPCC.2000.882463.

Winter, D. A. (2009). *Biomechanics and Motor Control of Human Movement*. John Wiley Sons. DOI: 10.1002/9780470549148.

Yahyavi, A. et al. (2013). "Watchmen: Scalable Cheat-Resistant Support for Distributed Multi-player Online Games". In: *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pp. 134–144. DOI: 10.1109/icdcs.2013.62.