Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 1

Simulating Multijoint Upper Limb Movement in Multiple Dimensions Using the Equilibrium Point Control Approach

Jan Kerner

Course of Study:	Informatik
Examiner:	Prof. Dr. Andreas Bulling
Supervisor:	Dr. Daniel Haeufle
Commenced:	3. May 2021
Completed:	3. December 2021

Abstract

Finding equilibrium points makes it possible to control posture and movement, without the need of solving the inverse dynamic problem. They can then be used to simulate movement of healthy, as well as impaired biomechanical systems accurately. Not being limited by complexity or dimensionality is desirable for methods to find them, in order to achieve the most accurate results, also in complex environments. In this work, different methods for finding equilibrium points are investigated and applied to musculoskeletal models of increasing complexity, using the open source simulation software OpenSim. In sum, our results show that it is possible to find equilibrium points in multi-joint systems, without having to solve the inverse dynamics problem in a reasonable runtime, while still getting acceptable accuracy.

Zusammenfassung

Durch das Finden von equilibrium Punkten, ist es möglich Haltung und Bewegung kontrollieren zu können, ohne, dass inverse dynamik Problem Lösen zu müssen. Diese können dann benutzt werden, um gesunde, als auch gestörte biomeachnische Bewegungen zu simulieren. Nicht durch die Komplexität und die Dimensionalität des Problems eingeschränkt zu sein, ist erstrebenswert für unsere Methoden, um die genausten Ergebnise erzielen zu können, auch in komplexeren Umgebungen. In dieser Arbeit, werden verschiedene Methoden, equilibrium Punkte zu finden, untersucht und im Nachhinein an Muskel-Skelett Modellen, mit steigender Komplexität, ausgetestet. Dies wird mithilfe, der open-source Software OpenSim umgesetzt. Zusammenfassend, zeigen unsere Ergebnise, dass es möglich ist, equilibrium Punkte in mehrgelenkigen Systemen zu finden, ohne das inverse dynamik Problem lösen zu müssen und gleichzeitig eine akzeptable Laufzeit mit akzeptablen Ergebnisen zu generieren.

Contents

1 Introduction

Being able to simulate biomechanical movement as accurate and reliable as possible in a computationally cheap way, gives us the possibility to investigate its implications in reasonable time and without the need of a test subject. This also leads to a better understanding of movement and can be applied to a many different research topics, such as computer science, medicine, robotics and biology. Since there is an increasing number of people with movement disorders in the future [2], because of the rapidly aging population the importance and demand of assistive devices for counteracting them, will increase. Having the ability to simulate these disorders accurately, will make it possible to investigate the causes as well as making the whole designing and testing process for different assistive devices easier and thereby more efficient. It can even be possible, to reach a state of research in the future, such that every person can increase their quality of life with these devices, without having to suffer from this type of motor control impairment anymore [7] [18]. But due to the high complexity of movement in our three dimensional world, we are still on the lookout for methods to decrease the runtime and increase the accuracy for these approaches. To simulate a movement system, we have to think about a suitable level of detail, regarding our biomechanical models, with enough information gain, to analyse the properties of the motor control system, but also being able to simulate it on a computer. With the use of today's computers and already provided algorithms for solving differential equations and given simulation environments, we are already able to solve complex systems of differential equations in a reasonable runtime[15]. There are a lot of different approaches to simulate and create movement, such as motion capturing [13] or various inverse dynamic approaches[11]. However, we choose to use the equilibrium point control approach, because it is a promising forward dynamic method for solving single joint movement[9] and extend it to multiple joints. Our methods will then be tested with upper limb musculoskeletal models. To create movement, the model has to traverse through these equilibrium points. But before even being able to traverse through them, they first have to be found.
Therefore our goal of this work is to provide algorithms, for finding equilibrium points, inside musculoskeletal models. They can be then used to create movement through them. Furthermore it will help to make investigations of movement, even in a complex manner, easier, faster and more accurate. The goal in the future, will be to use this solution to simplify the evaluation and designing for assistive devices, to make the whole research process faster and cheaper.

2 Foundations

Simulating a system, requires computational power. But since our capabilities are limited, we are not able to compute everything down to its tiniest details. Therefore, we have to create models, to simplify and abstract different properties or even erase these, which are not needed. In the following, we will describe how this is done and which simulation environment is used.

2.1 Musculoskeletal Models

Starting with the abstraction of movement, we will use musculoskeletal models. They are systems of muscles and bones, which combines the complex human muscle structure to similar force directions to be able to simulate them. Here the musculoskeletal models consist of two parts, the skeletal part and the muscular part.

Skeletal Part

The skeletal part consists of rigid bodies and joints. In addition to them, the skeletal part also holds the spatial properties, like degrees of freedom, rotation, center of mass and muscle attachment points.

Muscular Part

The muscular part consists of the muscles of the model and the tendons, as well as their muscle properties. The muscles used here are Hill-Type muscles[8], which are the standard models in biomechanics. They roughly consist of two parts. The contractile part, which represents the nonlinear muscle part and the elastic part, which represents the tendon part. They are described by 2 ordinary differential equations. Important to note here is that these models do not have a mass or inertia. These properties are already accounted for in the skeletal part. Every muscle is driven by one α-motor neuron, which controls, how much the muscle contracts. In the following the intensity of these neurons is between [0, 1], where 1 means that the muscle contracts at it's maximum force and 0 that there is no muscle activation.

2.1.1 Simple Model

The Simple model shown in Figure 2.1 is a musculoskeletal model with 2 muscles and 1 degrees of freedom. The two muscles function as flexor and extensor. Because of the low complexity of this model, it is possible to use it as a initial platform for testing different approach ideas and achieving quick results, without spending to much time on problems like multi dimensionality and multiple joints.

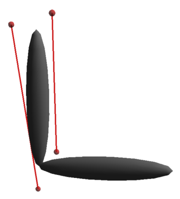

Figure 2.1: Simple Model consisting of 2 muscles, 2 rigid bodies and 1 degree of freedom

2.1.2 Arm 26 Model

The Arm26 model as shown in Figure 2.2 is a right upper extremity musculoskeletal model with 2 degrees of freedom, actuated by 6 muscles and is used as a more complex expansion for the previous simple model. Due to the higher amount of muscles and joints, we now have to consider the scalability and runtime of our algorithms more.
New Problems: Multiple joints

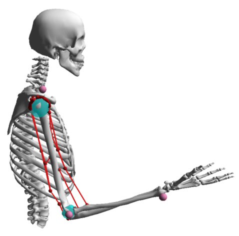

Figure 2.2: Arm26 model consisting of 6 muscles and 2 degrees of freedom

2.1.3 MOBL Arms Model

The MOBL arms model as shown in Figure 2.3 it consists of 50 muscles and has 7 degrees of freedom [12], [10]. This model is used as the most complex model and adds, besides of the higher amount of muscles, also one more dimension, since movement with this

model is now possible within all three dimensions.
New Problems: Multiple joints, more muscles and three dimensions

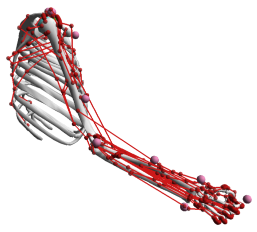

Figure 2.3: MOBL Arms model consisting of 50 muscles and 7 degree of freedom

2.2 Equilibrium Point

Knowing how our models are built up, we will discuss how to create suitable movement in the following.

An equilibrium point, also equilibrium position, is a steady state of the system[4], which can be achieved due to the antagonistic muscle setup and the muscles force-length relation. In our case, if the torque on the joints are zero. This characteristic is also called open-loop stable. Open-loop stability means that, depending on the muscles activation signals, the antagonistic muscle setup, will always equilibrate into a equilibrium position. As seen in Figure 2.4, each equilibrium position can be reached with different combinations of muscle activations. The only difference is that the joint stiffness changes. Applying an external force to these equilibrium positions results in viscoelastic restoring forces, the so called preflex forces, which are counteracting the applied forces.

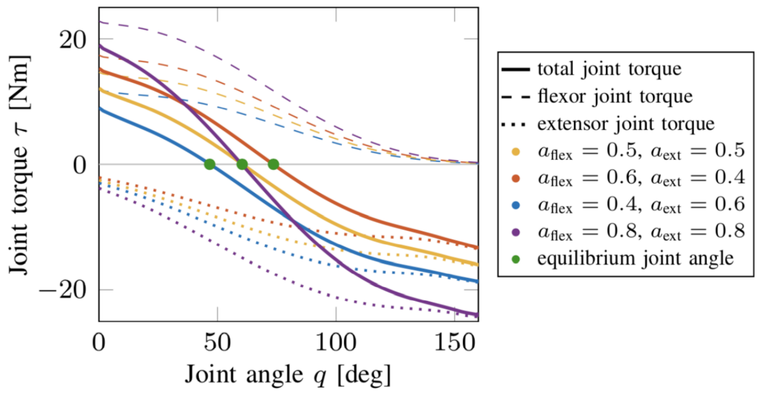

Figure 2.4: This figure shows an example of an antagonistic muscle pair, consisting of a flexor(flex) muscle and an extensor(ext) muscle and in which positions it will equilibrate by applying different muscle activation signals a to them. The equilibrium positions are marked by the green dots and the slope of the line, crossing them, indicates the stiffness. Higher slope, means higher stiffness and vice versa.

2.2.1 Equilibrium Point Control

Equilibrium point control describes a motor control action. By switching from an equilibrium point to another one, the movement is underlying a movement, which minimizes the muscle activity [3].

2.3 Simulation Environment

As simulation environment, we are using OpenSim, which is open source and is used for modeling and simulating dynamic movement of musculoskeletal models[5], [14]. Controlling the environment is done by using the C++ SDK, provided by OpenSim.

2.3.1 Defining a state

Since our simulation environment does not give us the possibility to step through every single simulation step, in fact it simulates a given time period, we need to define a constant state in this continuous environment.

Therefore a constant state is defined as state of zero movement. To be more precise, if there is no movement in every joint. But since we are using muscles, the system underlies a damped oscillation. This can lead to a long runtime, until the system comes to a halt. To decrease the runtime, we added an upper and lower bound, in which movement is still allowed.
A constant state is then reached, as soon as the angle of a joint at timesteps $n-k \dots n-1$ does not exceed the angle bounds of that joint at timestep n, where n denotes the timestep for this constant state and k the amount of timesteps, before n was reached. This has to hold for every joint.

The Algorithm 2.1 takes as input a two dimensional of size $j \times k$, where j denotes the number of joints.

Algorithm 2.1 Check for zero movement

```
procedure isStableState($timesteps[][], intervall\_range$)
    delta ← intervall_range
    check ← True            // never changes to false, if every angle is inside the bounds
    for $l_1 \in \{0 \dots j-1\}$ do                                  // loop through joints
        for $l_2 \in \{0 \dots k-1\}$ do                          // loop through last k steps
            if NOT(timesteps[n − 1][$l_1$] − delta <
                timesteps[$l_2$][$l_1$] < timesteps[n − 1][$l_1$] + delta) then
                check ← False               // if the angle is out of bounds, set to false
                break
            end if
        end for
    end for
    return check
end procedure
```

2.3.2 Defining the error function

We now know how to define a state in our environment. The next important step is now to define a function, which can tell us something about the quality of our state. In other words, what is the "difference" or "error" between the current state and the goal state. To achieve that, we will define an error function.

Let $\{j_1, \dots, j_n\}$ with $n \in \mathbb{N}_{>0}$ be the set of joints and $\{\phi_{j_1}, \dots, \phi_{j_n}\} \in [0; 360)$ the set of corresponding joint angles. Also let $\phi^g_{j_i}$, with $i \in \{0, \dots, n\}$ denote the goal angle of joint i.

Our error function is then defined as

$$Err(j_1, \dots, j_n) = \sum_{i=1}^{n} |\phi_{j_i} - \phi^g_{j_i}|.$$

2.3.3 Finding Equilibrium Points

After defining a state and its distance to the goal state, we now come to the point, where we have to define a goal state. Since we are searching for equilibrium points, our goal state is defined as a state, in which the joint angles are as close to the desired joint angles of the equilibrium point as possible.

This implies that a goal state is reached, as soon as the error function gets minimized.

In Algorithm 2.2, this goal state will be reached, if the error can not be decreased any further.

Algorithm 2.2 check for goal state

```
procedure ISGOAL($best\_angles[], goal\_angles[]$)
    check ← True
    for move ∈ possible_movements do
        current_angles = $simulate(move)$
        if error_function_decreased($current\_angles, best\_angles, goal\_angles$) then
            check ← False
        end if
    end for
    return check
end procedure
```

3 Related Work

Equilibrium point control was first introduced in a series of papers from Feldman and his colleagues [6], [1], who investigated the basic rules of the underlying motor control system. Later it was shown by Kistenmaker et. al, that it is indeed possible by using these rules to realize fast movement with single joint models[9].

Previous work was already done by Katrin Stollenmaier and their colleagues, who were investigating the compensation of motor control deficits [17]. They used the Arm26 model [3], to simulate impaired movement and applied assistive force in different ways to neutralize perturbation. This movement was also made with equilibrium point control. Unlike our approach, they determined the muscle activation values are by minimizing the sum of the difference between muscle stimulations and the desired level of co-contraction at each equilibrium position. Based on their findings, they were able to make predictions on the design of assisitve devices. Similar and based on the previous work the team of Katrin Stollenmaier, also designed a testbed, called ATARO for investigating different motor control techniques. ATARO is a robotic arm, which is based on a numerical neuro-musculoskeletal model of vertical arm movement in the sagital plane [16]. However these approaches were made in a two-dimensional plane. Our approach is now to aim for arbitrary movement in a three-dimensional world. This would make it possible to compare our results to them, already obtained in a two dimensional plane, as well as discovering new techniques for impairment control.

4 Approach

With the gathered knowledge about musculoskeletal models and their behaviour, we are now able to define our problem, to find a suitable solving mechanism afterwards.

Since finding equilibrium points of a model g with m muscles and j joints, comes down to an optimization problem, we can describe it as follows.

$$g(a_1, \dots, a_m) = (\phi_1, \dots, \phi_j),$$

where $a_i \in [0,1], i \in \{1, \dots, m\}$ be the muscle activations and $\phi_k \in [0, 360), k \in \{1, \dots, j\}$ the joint angles.
To be able to find the correct muscle activations, we have then to minimize the Err(x) function, shown in Section 2.3.2

To be able to find these muscle activations according to the joint angles, we came up with different solutions, which differ in implementation design and efficiency.

4.1 Muscle Movement

Knowing how the muscles work in our simulation, forms the basis of the implementation design for our algorithms. We know that a muscle can contract or extend, depending if the muscle activity increases or decreases. By that we can change the muscle force according to the muscle activity. This allows us to steer the model and its movement.

4.2 Brute Force

We will now start to look at the development process and the different solving mechanisms, we came up with.

The first and simplest implementation idea is to make a small step into every possible movement direction from one initial position and test it. If we then find a position, which is closer to our desired position, we continue doing that for this better position

and so on, until we reached our goal.
Since we can only move the model by changing the muscle activations, the idea of checking every direction comes down to changing them in every possible way. To make a small step into every possible direction, we have to apply a small activation change to every possible muscle activation combination. We are basically brute forcing every direction, until we get closer to our goal.

The black part of Algorithm 4.1 shows the implementation of this idea.

With looking at Figure 4.1, we can already make a first improvement to our algorithm, by getting rid of the double checking problem.

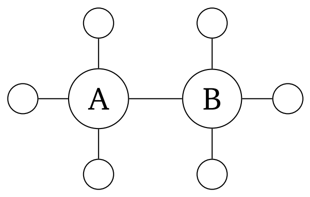

Figure 4.1: This graph shows the double checking problem. Starting from position A and choosing position B as best successor position, leads to the problem, that A has to be checked again, when checking the successor positions of position B, although we already know that A is a worse position than B

This is done by adding a cache into our Algorithm 4.1, which is highlighted in red and leads to an already improved runtime.

Algorithm 4.1 Brute Force

```
procedure FINDEQP($goal\_angles[]$)
    visited = $\emptyset$
    best_angles = initial_angles
    best_activations = initial_muscle_activations
    while True do
        best_result_changed = False
        combinations = get_muscle_combinations($muscle\_count, best\_activations, step\_size$)
        for muscle_combination $\in$ combinations do
            if muscle_combination $\in$ visited then
                continue
            end if
            set_muscle_activity($muscle\_combination$)
            joint_angles = simulate_model()
            visited.add($muscle\_combination$)
            if error_decreased($joint\_angles, best\_angles$) then
                best_angles = joint_angles
                best_activation = muscle_combination
                best_result_changed = True
            end if
            if best_result_changed == False then
                break
            end if
        end for
    end while
    return ($best\_activations, best\_angles$)
end procedure
```

As shown before, adding a cache with already visited muscle activations is important to decrease the runtime. But even with a cache, this algorithms still leaves us with an exponential runtime.

4.2.1 Stochastic Approach

Since the runtime complexity of the brute force approach is high, it is not a suitable solution for more complex models. Therefore, instead of trying every possible movement direction, we limited the search space per step, by a fixed number of random directions. This is done by replacing the **get_muscle_combinations** function in Algorithm 4.1 with the function called **get_random_muscle_combinations**

4.3 Pattern Search

We solved the problem of the exponential runtime, by adding a stochastic behaviour to our algorithm. The remaining problem is now, that, although we managed to improve our runtime, a good result is not guaranteed, since our random approach can pick directions in a bad, or non direct way.

Algorithm 4.2 Pattern Search

```
procedure PATTERNSEARCH($goal\_angles[], accuracy$)
    combinations = []
    goal = $False$                                  // checks, if the goal state is reached
    step_size = $0.1$
    best_activations ← initial_muscle_activations
    best_angles ← initial_angles
    while NOT goal do
        better = $False$                            // checks, if a better state was reached
        combinations = get_search_combinations(best_activations, step_size)
        for muscle_activity ∈ combinations do
            set_muscle_activity($muscle\_combination$)
            joint_angles ← simulate_model()
            if error_decreased($joint\_angles, best\_angles$) then
                best_angles ← joint_angles
                best_activations ← muscle_combination
                best_result_changed ← True
                better = $True$
                break                   // break loop, since better position was found
            end if
            if NOT better AND step_size < accuracy then
                return ($best\_activations, best\_angles$)
            end if
        end for
        if better then
            step_size∗ = 2
        else
            step_size/ = 2
        end if
    end while
    return ($best\_activations, best\_angles$)
end procedure
```

The idea of pattern search is, that we are comparing the movement, of each muscle change one by one. We then take the state, which is closest to our goal state. The advantage of this approach is, that we are able to adapt our step size. More precisely, this means, that our search space will be very wide in the beginning and becomes more accurate, the closer we approach the goal state.

4.4 Iterative Solution

In the previous approaches, we tried to find our goal position by changing all joint angles at once. The goal of this implementation is to set the joint angles one by one. If the adjustment of a joint n changes the already set joints $1, \ldots, n-1$ by more than a threshold t, these joints will then be brought back in place, until all joints are at their desired angle. For finding the single angles, we will use the already discussed brute force or pattern search algorithm, but with the difference, that they are only applied to one joint at a time and stop their execution, as soon as the other joints moved out their position too much, which is done by the **perform_simulation_step** function.

The implementation of this idea is shown in Algorithm 4.3. This procedure takes as input the algorithm, with which we want to search for the joint angles and the desired goal angles.

Algorithm 4.3 Iterative Solution

```
procedure ITERATIVESEARCH($goal\_angles[], algorithm$)
    current_joint_index = $0$
    best_angles[] = get_joint_angles()
    goal = False                                   // checks, if the goal state is reached
    while NOT goal do
        joint_goal = $True$                        // checks, if the goal state of one joint is reached
        perform_simulation_step($algorithm, current\_joint\_index$)
        joint_angles = get_joint_angles()
        if error_decreased($joint\_angles[current\_joint\_index]$,
                    $best\_angles[current\_joint\_index)]$ then
            joint_goal = $False$
            best_angles[$current\_joint\_index)$] = joint_angles[$current\_joint\_index$]
        end if
        if joint_goal == $True$ then
            current_joint_index+ = $1$
        end if
        for i $\in \{0, \dots,$ current_joint_index $- 1\}$ do       // if any previous angle changed
            if |goal_angles[i] − joint_angles[i]| > t then
                current_joint_index = i
            end if
        end for
        if current_joint_index > sizeof($goal\_angles$) then
            goal = $True$
        end if
    end while
    return(get_muscle_activations(), best_angles)
end procedure
```

5 Results

This section shows the results of our work on the different used models. Here, we will focus on the two main aspects of our algorithms. The time it takes to find an equilibrium point and how accurate our results can get. Finally, we will look at the runtime tradeoffs, we need to consider to achieve better results.

5.1 Previous Findings

Before investigating the results, a couple of previous findings have to be discussed.

Arm26 Model

During the experiments with the Arm26 model, it turned out, that the original OpenSim model had some issues and was not well conditioned for our task. We solved this, by manually rerouting the muscles attachment points of the musculoskeletal model. After these adjustments, it was possible to move the shoulder. However, there were still issues, because in some situations, the model reached an undefined state, or got stuck. Therefore, we had to search different starting and goal points manually, to ensure, that these states, could be reached. This prohibited us from searching for arbitrary equilibrium points. Nevertheless, we were still able to test our algorithms, although the equilibrium points, were chosen beforehand.

Iterative Solution

The iterative solution turned out to be unsuitable for our approach, since the adjustment of the joint angles, often lead to an infinite loop of alternating adaptation of the joint angles. This behavior occurred significantly more often than the completion of the algorithm. Therefore, we are not taking these results into consideration in this section.

MOBL Arms Model

Using the MOBL Arms Model in OpenSim was impossible. After starting the simulation, OpenSim took multiple seconds, just to simulate a fraction of a second, of the model itself, even if there were no muscle activations put onto the model yet. This lead to the decision, that it does not make sense to continue investigating this model any further. 25

5.2 Execution Time

At first, we will have a look at the execution time. As a tool for measuring, we used the c++ standard library chrono. By applying the **std::chrono::system_clock::now** function in front and after the search algorithms and afterwards taking the difference, we were able to calculate the runtime.

time[ms]	Brute Force (BF)		BF With Cache		Pattern Search (PS)
Model	Deterministic	Random	Deterministic	Random	
Simple	40246	39552	29331	31022	55218
Arm26	199429	199399	180818	204287	89753

Figure 5.1: Measurements were made with a step size of 0.001 and the execution times were averaged over 10 runs. For the simple model, a cache for the det. BF approach, decreased the runtime by 27%, and by 21% for the random approach. PS approach was 46% slower, than the fastest, the det. BF with cache, approach. For the Arm26 model the cache increased the runtime by 9% for the det. BF approach and is not signifikant for the random approach. PS was 50% faster than the previous fastest det. BF with cache approach

As seen in Figure 5.1 adding a cache, already decreased the runtime. This was expected, as already discussed in Chapter 4. Comparing the brute force approaches, the random approach took nearly the same as the deterministic approach for the simple model and much more for the Arm26 mode. This can be explained, due to the fact that the random approach does not try every possible movement direction and therefore takes not the "most direct" path to the goal state
Against our expectations, the pattern search algorithm for the simple model takes way more time than the brute force approach. This is based on the fact that this algorithm takes big jumps at the beginning, which also leads to longer simulation times, since the

joints have to be moved to these positions. But compared to the Arm26 model and the higher muscle amount, the runtime of the brute force algorithm rockets upward, while the pattern search algorithm becomes way more efficient than before.

5.3 Accuracy

In this part, we will discuss the accuracy of our different approaches. We will only investigate the accuracy of the pattern search algorithm and the deterministic brute force algorithm.

The accuracy for the simple model and for the Arm26 models turned out to be accurate, for smaller step sizes as seen in Figure 5.2 and Figure 5.3

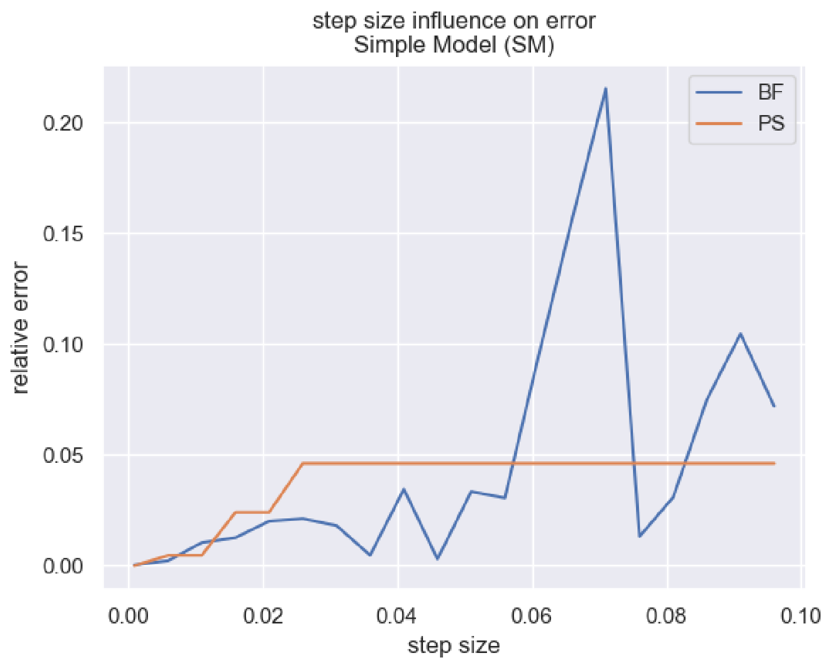

Figure 5.2: The relative error for the BF(blue) approach, increases very quick for higher step sizes and even hit 20% at its peak, while the relative error for the PS(orange) approach increases, but stays the same at some point.

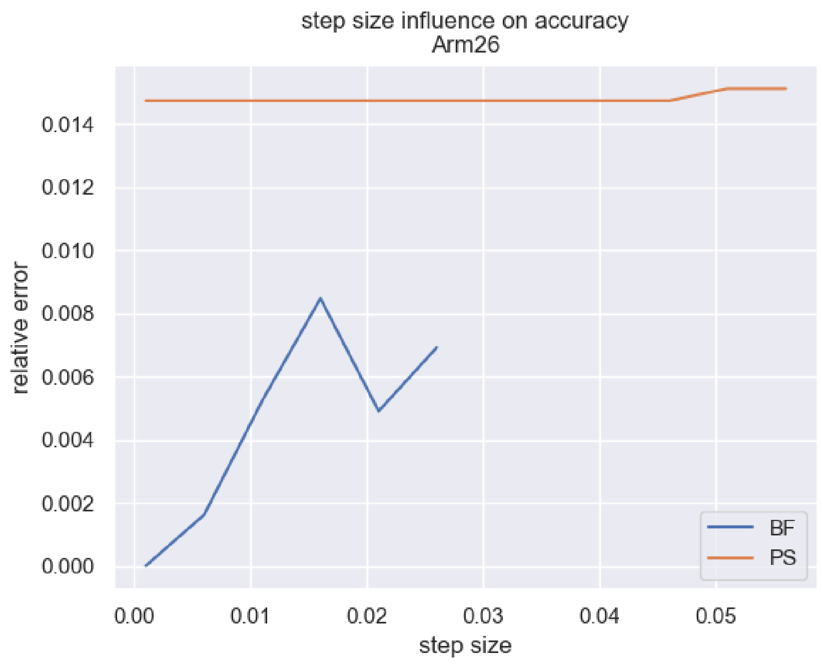

Figure 5.3: The relative error for the BF() approach, increases very quick for higher step sizes and even hit 20% at its peak, while the relative error for the PS approach increases, but stays the same at some point. Important to note here is that due to the fragile Arm26 model, the step sizes for the brute force approach could only be measured above a step size of 0.025.

5.4 Accuracy and Runtime

As we have already seen, a smaller step size leads to higher accuracy. In the following, we will also have a look at the influence the runtime is affected if we want to achieve more accurate results. Also here we will only investigate the accuracy of the pattern search algorithm and the deterministic brute force algorithm

By looking at the graphs in Figure 5.4 and Figure 5.5 we can observe, that the runtime increases, as soon as the step size decreases, for higher accuracy.

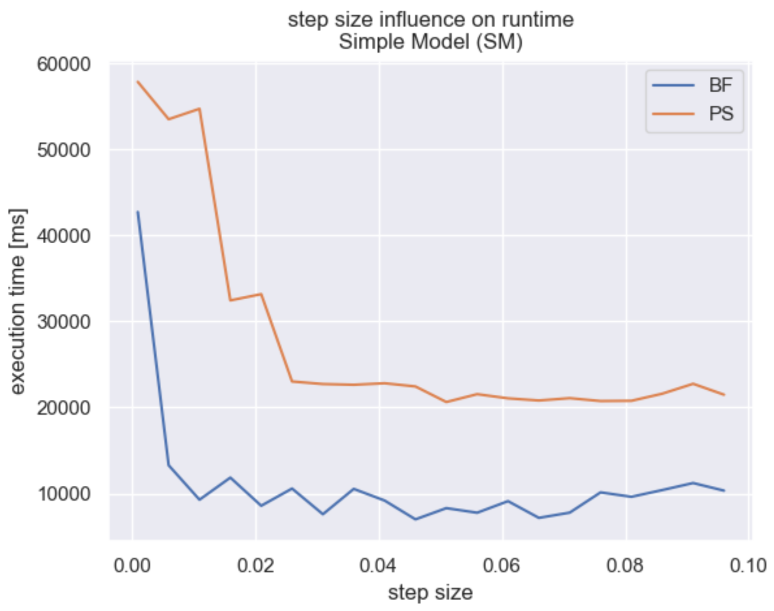

Figure 5.4: The runtime of both algorithms decreased for smaller step sizes. An interesting observation is that the runtime is not decreasing linearly, it more or less stays the same at some point.

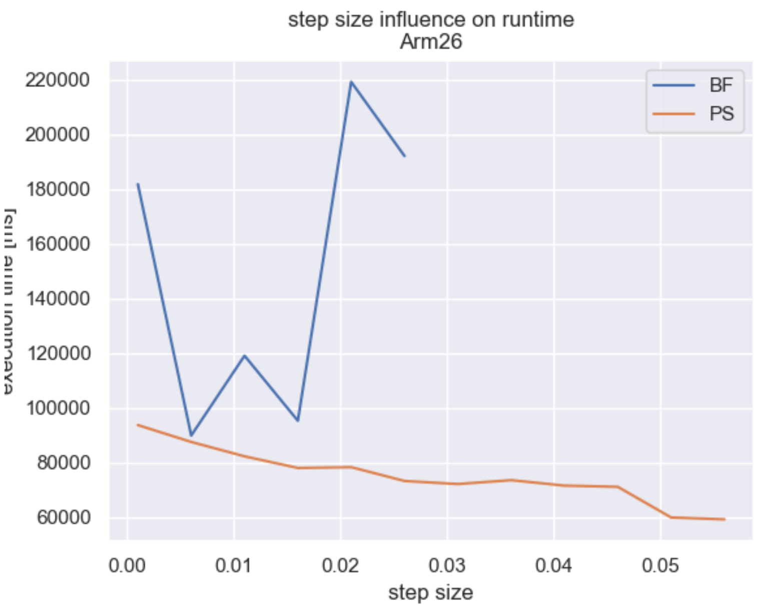

Figure 5.5: The runtime for the PS approach decreases with higher step sizes monotonically. The runtime for the BF approach also seems to decrease, but due to the instability of the model, we can not make assumptions here.

5.5 Initial Position and Runtime

Starting with a good initial position leads to the fact that the distance to the goal angle and therefore the error function, is already small from the beginning. In this section, we will compare different initial positions and their importance on the runtime.

Important to note here is, that we are only looking at the results from the simple model, since, as already explained in Section 5.1, we were not able to use arbitrary equilibrium points.

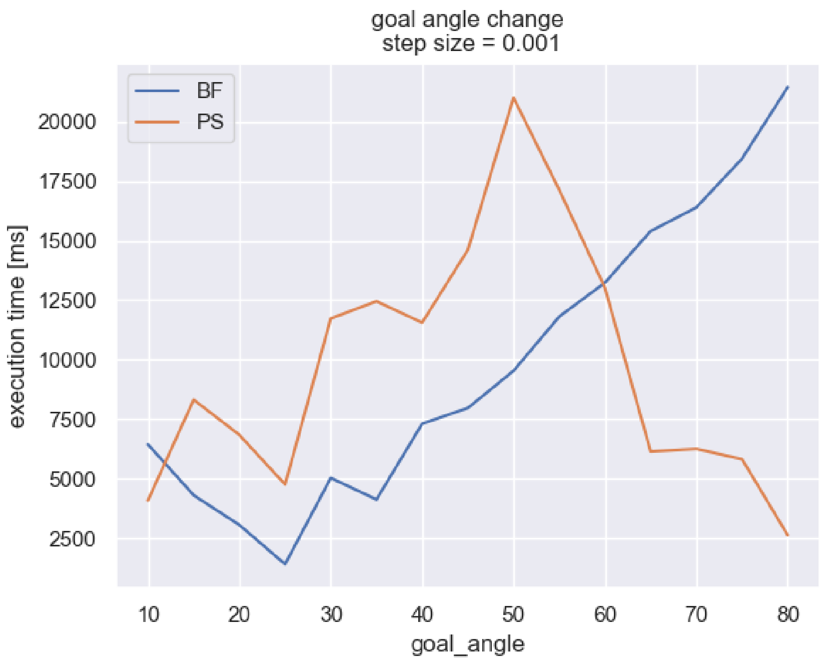

Figure 5.6: The initial position was at 45°. The brute force(BF) approach, shown at the blue line forms the shape of an "U", meaning it needs more time to find an goal angle, which is further away. The pattern search (PS) approach, shown by the orange line, forms a "hat" which means, it needs more time to find goal angles, nearby.

An explanation for the behaviour of the pattern search algorithm shown in Figure 5.6 is that due to the higher step size at the beginning of the algorithm, it reaches angles, which are further away, much faster, than the brute force approach, which has a constant step size. This is also the explanation, why it needs more time to find equilibrium points, which are further away.

5.5.1 Introducing the Database

With the information we gathered from Section 5.5, we came up with the following idea. By saving already visited nodes inside a database, it is possible to query the closest equilibrium point and start the search from there. Assuming we then have a grid of equally distributed found equilibrium points with, we are then able to query the closest node to our goal node and start the search from there.

6 Conclusion

This work, presents various approaches to use and find equilibrium points for biomechanical motion control. Our shown results are revealing that our approach can find equilibrium points in reasonable runtime. The most promising result are yielded by the solution, which is using the pattern search approach. According to our results, we assume that it is also possible to handle even more complex models, which are not part of this work. Moreover, it should also work to find equilibrium points in musculoskeletal models, which are not limited to the upper limb. We expect, that our approach is also relevant for any equilibrium point control based approach, used for simulating movement. It is possible to use our findings for acquiring equilibrium points in order to create movement.

6.1 Limitations

6.1.1 Algorithms

The algorithms are showing promising results, but are not tested in three dimensional space yet. Nevertheless, our approaches can be theoretically also used in higher dimensions, since they are not limited to any dimension.

6.2 Future Work

Although our approaches are already yielding reasonable results, there is still some space for improvement in some aspects.

Better conditioned model

Because of the poor condition of the Arm26 model, we had to limit the muscle activations, so it does not behave in an unexpected way. Testing the algorithms with better conditioned models or even models, which are not limited to the upper limb or even for a universal usage, would be desirable.

Increase performance

One of the biggest problems, we had during our research was the poor performance of OpenSim. This lead to a long waiting time, while executing our tests as well as not being able to simulate models, with way higher complexity than the tested Arm26 model. By switching from OpenSim to MuJoCo, we are expecting a significantly decreasing runtime, since MuJoCo is shown to be 600 times faster than OpenSim.

More Complex Musculoskeletal Models

The next step in testing the functionality of the algorithms would be to use and test them on a three dimensional model, with more muscles than the Arm26 model, such as the MOBL Arms model. In additions we could also advance from upper extremity models, to arbitrary musculoskeletal models.

Add Constraints

Our algorithms are suitable for finding equilibrium points, but since equilibrium points are not unique and differ in stiffness, the final result depends on the initial state. Adding constraints for stiffness, would also offer the possibility to influence this variable.

Inverse Kinematics

Another, not tested, approach for finding equilibrium points would be to use inverse kinematics. In our forward approaches, we always tried to start from an initial positions and start searching for the equilibrium point, until we reached it. The idea of the inverse kinematics approach is to set the rigid bodies to a desired position and fix them in this position. After fixing them, the muscles are still applying force to the joints, but without movement. The goal is now to find muscle activities, such that the torque at the joints will be 0. Releasing the fixed rigid bodies, will then result in the same position, since the muscles are now in their equilibrium state.

6.3 Acknowledgement

I want to give a very special thank to my supervisor Daniel Häufle, who was always within my reach if, I ever had some problems and open questions. I would also like to thank Pierre Schumacher and Fabio Izzi, which were part of this research group. All of them supported me through the whole thesis and even in times of ignorance, they always had some good ideas to grab onto and were helping me out with their already gained expertise in this research field. Without their help and without the help of my always encouraging friends and family, this whole thesis would have been quiet more difficult.

Bibliography

[1] D. Asatrian, A. Fel'dman. "On the functional structure of the nervous system during movement control or preservation of a stationary posture. I. Mechanographic analysis of the action of a joint during the performance of a postural task." In: *Biofizika* 10.5 (1965), pp. 837–846 (cit. on p. 15).

[2] J.-P. Bach, U. Ziegler, G. Deuschl, R. Dodel, G. Doblhammer-Reiter. "Projected numbers of people with movement disorders in the years 2030 and 2050." In: *Movement disorders* 26.12 (2011), pp. 2286–2290 (cit. on p. 7).

[3] A. Bayer, S. Schmitt, M. Günther, D. Haeufle. "The influence of biophysical muscle properties on simulating fast human arm movements." In: *Computer methods in biomechanics and biomedical engineering* 20.8 (2017), pp. 803–821 (cit. on pp. 12, 15).

[4] "Equilibrium Point." In: *Encyclopedia of Neuroscience*. Ed. by M. D. Binder, N. Hirokawa, U. Windhorst. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1145–1145. ISBN: 978-3-540-29678-2. DOI: 10.1007/978-3-540-29678-2_3074. URL: https://doi.org/10.1007/978-3-540-29678-2_3074 (cit. on p. 11).

[5] S. L. Delp, F. C. Anderson, A. S. Arnold, P. Loan, A. Habib, C. T. John, E. Guendelman, D. G. Thelen. "OpenSim: open-source software to create and analyze dynamic simulations of movement." In: *IEEE transactions on biomedical engineering* 54.11 (2007), pp. 1940–1950 (cit. on p. 12).

[6] A. Fel'dman. "On functional tuning of nervous system during controlled or preservation of stationary pose. 3. Mechanographic analysis of human performance of simple movement tasks." In: *Biofizika* 11.4 (1966), pp. 667–675 (cit. on p. 15).

[7] A. Frisoli. "Exoskeletons for upper limb rehabilitation." In: *Rehabilitation Robotics*. Elsevier, 2018, pp. 75–87 (cit. on p. 7).

[8] D. Haeufle, M. Günther, A. Bayer, S. Schmitt. "Hill-type muscle model with serial damping and eccentric force–velocity relation." In: *Journal of biomechanics* 47.6 (2014), pp. 1531–1536 (cit. on p. 9).

[9] D. A. Kistemaker, A. (J. Van Soest, M. F. Bobbert. "Is equilibrium point control feasible for fast goal-directed single-joint movements?" In: *Journal of Neurophysiology* 95.5 (2006), pp. 2898–2912 (cit. on pp. 7, 15).

[10] D. C. McFarland, E. M. McCain, M. N. Poppo, K. R. Saul. “Spatial Dependency of Glenohumeral Joint Stability During Dynamic Unimanual and Bimanual Pushing and Pulling.” In: *Journal of biomechanical engineering* 141.5 (2019) (cit. on p. 10).

[11] C. Pizzolato, M. Reggiani, L. Modenese, D. Lloyd. “Real-time inverse kinematics and inverse dynamics for lower limb applications using OpenSim.” In: *Computer methods in biomechanics and biomedical engineering* 20.4 (2017), pp. 436–445 (cit. on p. 7).

[12] K. R. Saul, X. Hu, C. M. Goehler, M. E. Vidt, M. Daly, A. Velisar, W. M. Murray. “Benchmarking of dynamic simulation predictions in two software platforms using an upper limb musculoskeletal model.” In: *Computer methods in biomechanics and biomedical engineering* 18.13 (2015), pp. 1445–1458 (cit. on p. 10).

[13] A. Seth. “A Motion Tracking Method for the Modeling and Simulation of Human Movement in 3-D.” In: *ISB Dissertation Grant. University of Texas (2004-05 continuing Fellowship), NASA Grant (# NNJ04HI99G, PI: Dr. Marcus Pandy)* (2004), pp. 1–5 (cit. on p. 7).

[14] A. Seth, J. L. Hicks, T. K. Uchida, A. Habib, C. L. Dembia, J. J. Dunne, C. F. Ong, M. S. DeMers, A. Rajagopal, M. Millard, et al. “OpenSim: Simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement.” In: *PLoS computational biology* 14.7 (2018), e1006223 (cit. on p. 12).

[15] H. J. Stetter et al. *Analysis of discretization methods for ordinary differential equations*. Vol. 23. Springer, 1973 (cit. on p. 7).

[16] K. Stollenmaier, T. Nadler, C. Pley, W. Ilg, S. Wolfen, S. Schmitt, D. F. Haeufle. “ATARO: a muscle-driven biorobotic arm to investigate healthy and impaired motor control.” In: (2006) (cit. on p. 15).

[17] K. Stollenmaier, I. S. Rist, F. Izzi, D. F. Haeufle. “Simulating the response of a neuro-musculoskeletal model to assistive forces: implications for the design of wearables compensating for motor control deficits.” In: *2020 8th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics (BioRob)*. IEEE. 2020, pp. 779–784 (cit. on p. 15).

[18] R. J. Varghese, D. Freer, F. Deligianni, J. Liu, G.-Z. Yang, R. Tong. “Wearable robotics for upper-limb rehabilitation and assistance: A review of the state-of-the-art challenges and future research.” In: *Wearable technology in medicine and health care*. Elsevier, 2018, p. 340 (cit. on p. 7).

All links were last followed on November 11, 2021.

Declaration