



DEGREE PROJECT IN MECHANICAL ENGINEERING
SECOND CYCLE, 30 CREDITS

Motion tracking for virtual reality using inertial motion sensors

MAX LINDGREN

MARCUS OLSSON



KTH Industriell teknik
och management

Examensarbete TRITA-ITM-EX 2023:641

Motion tracking för virtuell verklighet med hjälp av IMU-sensorer

Max Lindgren

Marcus Olsson

Godkänt 2024-01-31	Examinator Hans Johansson	Handledare Fredrik Asplund
	Uppdragsgivare	Kontaktperson

Sammanfattning

Motion tracking är något som fram till nyligen varit dominerat av kamerabaserade system. Med framsteg i mikroelektromekaniska system (MEMS) så har nya lösningar baserade på IMU-sensorer börjat utforskas.

Detta examensarbete utforskar möjligheten att augmentera motion tracking för Virtuell Verklighet (VR) med IMUer. Mer specifikt så utforskas en befintlig metod, utvecklad för applikation inom skaderehabilitering, och evaluerar dess aptitud för en realskilda applikation.

Metoden implementerades och utforskades först för ideala förhållanden med hjälp av simulerad sensordata. Störningar introducerades sedan för utforska metodens robusthet och fallenhet för applikation inom VR. Hur stora och vilka typer av störningar som testades valdes baserat på prestandan av "köpa från hyllan" Commercial-Off-The-Shelf (COTS) IMUer. En rad olika rörelser simulerades för att utforska metodens beroende av gravitation som en dominerande del av accelerationen mätt av IMUerna, och också känsligheten för mätfel när en accelerationvektor som används i beräkningen av vinkeln är nästan kollinjär med ledaxeln. Utöver detta utforskades beräkningstiden genom att implementera algoritmen i ett lågnivåspråk och evaluering gjordes på relevant datorhårdvara.

Slutsatsen är att COTS IMUers prestanda är tillräcklig för applikationen och algoritmen kan köras snabbt nog på all hårdvara som är relevant för VR. Den största bidragande orsaken till fördöjning från det att sensorn registrerar en rörelse, till det att en vinkel är beräknad, är dataflyttning och numerisk derivering. Dock visades det att metoden producerar betydande fel i acceleration-baserade vinkelberäkningen när accelerationsvektorn som används avviker för långt från att vara vinkelrät mot ledaxeln. Slutsatsen är att metoden inte är lämplig för att spåra vinkeln på en armbåge eftersom armen kan röra sig väldigt fritt vilket medför att ledaxeln kan bli parallell med gravitationen.



KTH Industrial Engineering
and Management

Motion tracking for virtual reality using inertial motion sensors

Max Lindgren

Marcus Olsson

Approved 2024-01-31	Examiner Hans Johansson	Supervisor Fredrik Asplund
	Commissioner	Contact person

Abstract

Human motion capture has until recently been dominated by camera-based solutions. With advancements within Micro Electro-Mechanical Systems (MEMS) technology new Inertial Measurement Unit (IMU) based solutions are being explored.

This thesis explores the possibility of using IMUs for augmenting motion tracking for Virtual Reality (VR). More specifically it aims to explore one existing tracking method, developed for medical rehabilitation, and evaluate its suitability for tracking the angle of an elbow in real time.

The method was first implemented and explored in simulation with ideal sensor outputs. Disturbances were then introduced to explore its feasibility for VR. The magnitude and types of disturbances were chosen based on the performance of Commercial Off-The-Shelf (COTS) IMUs as well as the VR application. A range of motions was simulated to explore the methods stated reliance on gravity being a dominant part of the acceleration measured by the IMUs, and also the stated sensitivity to measurement errors when an acceleration vector used in the calculation of the angle is almost collinear with the joint axes. Additionally the computation speed was explored by implementing the same algorithm in a low-level programming language, evaluation was then done on relevant computer hardware.

It is found that COTS IMUs performance is sufficient for the application and the algorithm runs more than fast enough on any computer that is able to run VR. The major causes of delay between sampling the sensors and having a calculated joint angle is data transfer and numerical derivation. However it was found that the method produces significant errors in the acceleration based part of the angle calculation whenever the acceleration vector used strays too far from being perpendicular to the joint axes. It is concluded that the method is not suitable for tracking the angle of an elbow since the very free movement of the arm often puts the joint axes much too close to being parallel with gravity.

Contents

1	Introduction	5
1.1	Background	5
1.2	Research Questions	5
1.3	Ethical and Sustainability considerations	5
1.4	Delimitations	6
2	State of the art	7
2.1	Motion tracking in virtual reality	7
2.2	Motion tracking using IMUs	7
2.2.1	Hardware	7
2.2.2	IMU	7
2.2.3	Sensor fusion algorithms	7
2.2.4	Latency	8
2.2.5	Deep learning	8
3	Theoretical Background	9
3.1	The algorithm	9
4	Methodology	11
4.1	Case studies	11
4.2	The case	11
4.3	Setup	12
4.3.1	Replication	12
4.3.2	The VR case	12
4.4	Quality Assurance	12
4.4.1	Construct validity	12
4.4.2	Internal validity	12
4.4.3	External validity	13
4.4.4	Reliability	13
5	Implementation	14
5.1	The model	14
5.1.1	Starting with ideal gyroscope data	14
5.1.2	The Matlab script	15
5.1.3	Simulated motion	15
5.2	Adding noise to the IMUs	15
5.3	Evaluating latency	16
5.3.1	Data transfer	17
5.3.2	Algorithmic delay	17
5.3.3	Calculation time	17

6 Results	18
6.1 Verify the model	18
6.2 Simulation of noise	19
6.3 Resulting input lag	21
6.3.1 Data transfer delay	21
6.3.2 Numerical derivation delay	21
6.3.3 Calculation delay	22
6.4 Adding both noise and lag to the model	23
6.5 Testing different arm movements	24
6.5.1 Movement patterns	24
6.5.2 Movements in relation to gravity	25
7 Conclusions and future work	28
7.1 Conclusions	28
7.2 Future work	29
References	30
A C++ Implementation	32
A.1 Vec3.h	32
A.2 Vec3.cpp	33
A.3 Joint.h	35
A.4 Joint.cpp	36
A.5 MathFunctions.h	38
A.6 MockSensor.h	39
A.7 MockSensor.cpp	40
A.8 PerformanceTest.cpp	41
A.9 TestData.h	43

Nomenclature

COTS Commercial Off-The-Shelf

HMD Head Mounted Display

IMU Inertial Measurement Unit

LSB Least Significant Bit

MEMS Micro Electro-Mechanical Systems

RMSE Root Mean Square Error

SFA Sensor Fusion Algorithm

VR Virtual Reality

Chapter 1

Introduction

This chapter gives a background and a brief overview of the thesis.

1.1 Background

As Inertial Measurement Units (IMUs) are improving they have been considered and tested as an alternative sensor solution for human motion tracking. Today camera-based systems is the state of the art in this area. Human motion tracking is used in many applications such as sports training and analysis[1], rehabilitation medicine[2], ergonomics analysis[3], animated movie and game performances[4], science etc. Another area that has recently brought motion tracking to the consumer is Virtual Reality (VR). Here tracking is done and presented to the user in real time and currently also primarily using cameras and/or light-based sensors. Usually only three points are tracked, head and both hands. This makes it hard to portrait a compelling full body or even just arm representation. T. Seel, J. Raisch, and T. Schauer constructed a method for using kinematic constraints to estimate joint angle using IMU sensor data [5]. It will be further explored in this thesis for the case of tracking elbow angle for augmenting common VR tracking solutions. Tracking in VR requires both very high accuracy and tracking that works well in real time, without requiring too much computational time.

1.2 Research Questions

This project aims to reproduce the results of *IMU-based joint angle measurement for gait analysis*[5], using IMUs and kinematic constraints as a sensor solution and applying this to augment the tracking for VR. Thus allowing for more of the body to be accurately represented in the virtual environment. More specifically, the project has been divided into the following two research questions:

1. Can the results of [5] be reproduced?
2. If so, is the solution suitable tracking the angle of the elbow for application in VR; is the precision and update rate high enough to provide a robust and immersive solution for this application?

1.3 Ethical and Sustainability considerations

The algorithm and the simulations themselves do not have much impact on the environment. If used practically, the hardware for tracking is basically several IMUs attached to the body with wires in between.

The idea is to use the same computer as for the VR game itself to run the algorithms.

When it comes to ethics, the intention with this thesis is to increase the precision of body tracking for VR games and simulations using IMUs. However, in this thesis the main goal is to

show the concept using simulations. If the concept is used later practically, it has to be tested and verified that it increases the body tracking precision for VR in reality and that its consistent over time. This is to reduce the risk of virtual reality sickness rather than increasing it and also preventing the risk of human damage.

1.4 Delimitations

Joint type: The method this thesis aims to evaluate only works for joints with one rotational axis such as knee or elbow joints.

Targeted environment: VR is mainly used indoors, this research will be limited to evaluating use of the IMUs indoors.

Hardware: The system has to be as simple and cost-effective as possible to potentially reach a consumer market. The following delimitation are put on the evaluated hardware:

1. The IMU performance is limited to Consumer Off-The-Shelf(COTS) IMUs,
2. The algorithm will run on the same PC that is used for VR instead of being dependent on a separate PC or microcontroller.
3. The IMUs are assumed to be calibrated and have a known starting position and orientation.
4. The evaluation of the impact of noise will be limited to the internal noise in the IMUs and be based off specifications of COTS devices.

Chapter 2

State of the art

This chapter examines the state of the art of human body motion tracking.

2.1 Motion tracking in virtual reality

Currently most commercial VR solutions, like Oculus Quest [6] and HTC Vive [7] use cameras and other types of light sensors to track the motion of the user. So far the tracking has been simplified to only track three points, the head and each hand. Full body motion tracking can increase the immersiveness of the experience [8]. However, the current state of the art commercial solutions are too simple to provide it. The most similar forms of full body tracking solutions using cameras are far too complex and expensive for consumers[9].

An alternative solution for full body motion capture is to combine the current commercial VR solutions with IMUs. In recent years IMUs based solutions for body tracking has gained popularity. As a result, these suits have become cheaper and are getting closer to consumer price levels. For instance, a suit called Smartsuite Pro II from a company called Rokoko [10], and Mvn Link from Xsens [11].

2.2 Motion tracking using IMUs

In this section the state of the art of motion tracking using IMUs is examined.

2.2.1 Hardware

In terms of hardware, it was found that there are significant differences when comparing different commercially available IMU-based products [12]. This report presents the standard deviation of the measurements of the gyroscope, accelerometer and magnetometer respectively for each sensor package. While they all differ, none of them is superior across all three sensors.

2.2.2 IMU

An IMU is usually a combination of different sensors. The combination often used is a gyroscope, an accelerometer and a magnetometer. These sensors typically have 3 degrees of freedom each. There are different types of IMUs and one common one is called Micro Electro-Mechanical Systems (MEMS). MEMS-technology consumes little energy, is small in size and can be produced at a low cost. This report focuses on the gyroscope and the accelerometer due to the method explained in [5] which neglects to use the magnetometer. Unfortunately, datasheets for COTS IMUs are rarely giving a full view of the expected noise levels for the product.

2.2.3 Sensor fusion algorithms

Sensor Fusion Algorithms (SFAs), are algorithms that combine data from multiple sensors to provide a more accurate and reliable estimate of the state of a system. SFAs for IMU based

motion tracking is still an area of active research. In [12] ten different SFAs for orientation estimation are compared and no one best method is found. A reason for the diversity in methods is to reduce reliance on specific parts of the IMU depending on the goal of the algorithm. As mentioned in the article the performance of the accelerometer tends to deteriorate as acceleration increases. Likewise, magnetometers often lose precision because of surrounding electromagnetic disturbances. This is usually a problem for indoor applications due to nearby electrical equipment (TV, computer etc.). This issue is highlighted in many articles, for instance [12] and [13]. Without the magnetometer it is harder to calculate an absolute heading, i.e. in relation to earth's magnetic field. However, this is not necessary for tracking body movement. One workaround to avoid using the magnetometer is to introduce kinematic constraints on the model. For instance, by measuring the placement of IMUs in relation to the moving joint. The method evaluated in this report avoids using the magnetometer by leveraging more than one IMU and the natural constraints of the human body.

2.2.4 Latency

In the work presented in an article by O. G. R. Wittmann and F. Lamberg [14], upper body movement is tracked using IMUs which is then presented on a screen in front of the user. The measurement of end-to-end latency was done by having a high-speed camera filming both the sensor and the PC screen as a single scene. The latency was then calculated by visually comparing each frame of the film. The average end-to-end latency of the system, from sensor movement to a virtual body movement showed on a screen, was measured to 60.9 ± 7.7 ms. The commercial full body tracking system Mvn Link [11] is claimed to have a latency of 20 ms.

The article [15] endeavors to find how embodiment is affected by latency. The term embodiment is used in the context of VR to describe the sense for the user that the virtual avatar represents them, in a way where it starts to feel like their body. It was found that people are much more tolerant to latency in this aspect than when it comes to tracking of the Head Mounted Display (HMD) where increases in latency tend to lead to cybersickness. They found no significant loss in sense of embodiment until the latency is above 101 ms.

2.2.5 Deep learning

Recent developments have also been made in the field of deep learning. The model RIANN [16], and the models proposed by A. A. Golroudbari and M. H. Sabour [17] show real promise. They perform very well at the task of orientation estimation and according to the respective articles, they get significantly lower average errors than traditional SFAs. Neural networks like these can vary a lot in their computational complexity, however both methods claim to work for real time applications on modern hardware. Another benefit of these methods is that they do not require an initial resting period where the algorithm can converge. This is something that most methods developed before these required [17].

Chapter 3

Theoretical Background

In this chapter, the theory of the tracking method being studied is presented.

3.1 The algorithm

The model followed in this thesis [5] aims to produce the discrete angle of a hinge joint based on the output from two IMUs, one positioned on each side of the joint. The method combines two ways of producing the angle using sensor fusion, by way of a complementary filter. This is done to get the strengths of both methods and counteract their flaws.

The first method produces the angle by using the rotational rates from the gyroscopes of the IMUs. Then isolating the part of rotational rates that correspond to rotation around the joint and integrating over time to produce the angle.

$$\alpha_{gyr} = \int_0^t (g_1(\tau) \cdot j_1 - g_2(\tau) \cdot j_2) d\tau \quad (3.1)$$

Where $g_1(\tau)$ and $g_2(\tau)$ are the rotational rates of each IMU at a given time τ and j_1 and j_2 are the joint vector in the respective IMUs local coordinate system. This method is more resilient to noise but can drift over time.

The second method extracts the rotation also using the accelerations given by the IMUs. The radial and tangential acceleration due to rotation around the joint center is expressed by the equation below.

$$\Gamma_{g_i(t)}(o_i) := g_i(t) \times (g_i(t) \times o_i) + \dot{g}_i(t) \times o_i, i = 1, 2 \quad (3.2)$$

Here o_i is the coordinates of the joint center in local coordinates of each sensor. $\dot{g}_i(t)$ is the derivative of the rates of rotation and is calculated using a numerical second order approximation as seen below.

$$\dot{g}_i(t) \approx \frac{g_i(t - 2\Delta t) - 8g_i(t - \Delta t) + 8g_i(t + \Delta t) - g_i(t + 2\Delta t)}{12\Delta t}, i = 1, 2 \quad (3.3)$$

The accelerations $\Gamma_{g_i(t)}(o_i)$ are used in conjunction with the accelerations from the IMUs $a_i(t)$ to extract the acceleration of the joint center. This is done using the equation below.

$$\tilde{a}_i(t) = a_i(t) - \Gamma_{g_i(t)}(o_i), i = 1, 2 \quad (3.4)$$

Here $\tilde{a}_i(t)$ is the same acceleration expressed in the local coordinate system of each of the two IMUs. This means that the joint angle can be approximated by the angle between the projections of these two accelerations into the joint plane. In order to do this projection, two pairs of joint plane axes are defined as seen below.

$$x_1 = j_1 \times c, \quad y_1 = j_1 \times x_1, \quad x_2 = j_2 \times c, \quad y_2 = j_2 \times x_2, \quad c \nparallel j_1, \quad c \nparallel j_2 \quad (3.5)$$

Using these axes the approximation of the angle is described by:

$$\alpha_{acc}(t) = \triangleleft_{2d} \left(\begin{bmatrix} \tilde{a}_1(t) \cdot x_1 \\ \tilde{a}_1(t) \cdot y_1 \end{bmatrix}, \begin{bmatrix} \tilde{a}_2(t) \cdot x_2 \\ \tilde{a}_2(t) \cdot y_2 \end{bmatrix} \right) \quad (3.6)$$

$\triangleleft_{2d}()$ is the signed angle between two vectors in \mathbb{R}^2 . As no integration is used in this second method it is not susceptible to drift.

The two resulting angles from the two different methods are finally combined using sensor fusion. The work done in this thesis used the method described in [5], implemented as:

$$\alpha_{acc+gyr}(t) = \lambda \alpha_{acc}(t) + (1 - \lambda)(\alpha_{acc+gyr}(t - \Delta t) + \alpha_{gyr}(t) - \alpha_{gyr}(t - \Delta t)), \quad \lambda \in [0, 1] \quad (3.7)$$

Chapter 4

Methodology

This thesis aims to further study the method for motion tracking presented in [5]. The research method chosen to do this was a case study and the methodology is presented in this chapter.

4.1 Case studies

In the book *Case Studies in Software Engineering* by Per Runeson et al.[18] a software engineering case study is described as:

Case study in software engineering is an empirical enquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified.

They distinguish between four main purposes for doing research, exploratory, descriptive, explanatory and improving. They describe ways in which case studies can be used for other types of research but present it as primarily used for exploratory purposes, described as "*finding out what is happening, seeking new insights, and generating ideas and hypotheses for new research*". Collecting data for a case study can either be done qualitative, quantitative or a combination of the two aforementioned. After the data has been gathered in the right way, it also needs to be analyzed in a correct way. The gathered data needs to be validated in a case study to ensure the trustworthiness of the results. [18] divides validity into four different classifications, construct validity, internal validity, external validity, and reliability. Construct validity is to ensure the test measures the concept it was designed to evaluate. Internal validity is to ensure a causal relationship between for instance two variables. The relationship should stay the same even if it's influenced by other factors. External validity is the extent to which the findings in the study could be applied to other cases. Reliability is to ensure that the results can be reproduced under the same conditions. If for instance the same study is done by someone else, then the results should be the same.

4.2 The case

The purpose of this thesis is exploratory. The method being evaluated did not in its original report section [5] explore its feasibility for real-time applications. As described in 4.1 a case study is a natural choice for this type of exploratory research. The case chosen with the purpose of exploring this topic was VR. It is an application that puts a high demand on low latency and is primarily marketed towards end consumers which means the hardware must be cheap.

Commonly in VR applications the user is represented with their own virtual body. It is generally desirable that this virtual representation not only gives the user a sense of presence and agency, but even a sense of body ownership. As described in [15] the latency can have a large impact on these qualities.

As how much of the body is virtually represented is often limited to just the upper body or parts thereof, the choice was made to apply the tracking method to the elbow joint instead of the knee like in the original report [5]. It describes the tracking of the knee joint to have the benefit of being mostly constrained to movements that keeps the joint axis far away from being collinear with the acceleration vectors used in the calculation. The problem with this collinearity is described as: "*We shall note that the above equations are sensitive to measurement errors if the shifted accelerations $\tilde{a}_{1/2}(t)$, are almost collinear with the joint axes $j_{1/2}$* ". As the elbow joint is much less constrained the effect this has was further evaluated.

When it comes to price, it limits the choices in terms of available sensors. How the tracking method performs given the characteristics of cheaper IMUs should therefore be taken into account when looking at this case. For this case an IMU called SCL3300 from the company Murata was chosen, which is a COTS product based on the MEMS-technology [19]. It was chosen for this case due to its datasheet which gave some insights of expected noise levels which many other COTS IMUs did not show.

4.3 Setup

In this section the setup chosen for doing this case study will be explained.

4.3.1 Replication

In order to replicate the method from [5] an arm model was created in Simulink from which simulated ideal IMU sensor samples could be extracted along with a ground truth joint angle. The mathematical model was then recreated in MATLAB and tested using the simulated sensor samples.

4.3.2 The VR case

To evaluate the performance of the model for the VR case it was also tested with different types of noise applied. The types of noise and their magnitude was selected based of the characteristics of cheaper COTS IMUs. Furthermore, the latency of the method was evaluated and compared to what is found to be required for an experience of presence and embodiment.

4.4 Quality Assurance

In this section the steps that were taken to assure the quality of the study will be presented.

4.4.1 Construct validity

Reproduction of the results from the original report meant there was a clear reference to work from. In terms of new results, the very controlled simulated environment ensured tests could be constructed to look at specific effects of for example joint axes orientation or how dominant gravity is as a part of the measured acceleration.

4.4.2 Internal validity

The test setup using Matlab is beneficial for internal validity since the tests are conducted in a controlled environment with reliable data that is not affected by time. To ensure internal validity, different data sets, i.e. different arm movements, were tested. Tests were also made where the exact output from the simulation was combined with noise to confirm that the model was as robust to noise as described in [5]. The data sets were chosen to cover a wide range of arm movements. The root mean square error for all these data sets was extracted for comparison to the error reported in [5].

4.4.3 External validity

The degree to which the method examined in this thesis generalizes to arbitrary hinge joint tracking was examined. A wide range of movements was tested. The statement in [5] of the acceleration based estimation being more sensitive to measurement errors when the accelerations used in the calculation are "almost collinear" with the joint axes was closer examined, after the orientation of the joint being tracked was found to have a substantial effect on the angle estimation.

4.4.4 Reliability

The case study was conducted by producing a simulation model from which both ideal sensor samples and a ground truth joint angle could be extracted. With an exact ground truth to compare to, the reproduction of the original method from [5] could be clearly assessed. This result was then used as a baseline when applying noise characteristic to the IMUs relevant to this case. This way, exactly how the angle deviation changed when for example noise was applied, could be ascertained.

Chapter 5

Implementation

This chapter presents the arm model and the implementation of the algorithm presented in chapter 2. It was developed using MATLAB to simulate arm movements, calculate the elbow angle and present the results compared to the true angle retrieved from the model itself. This chapter also shows the reader the algorithm implemented using a low-level-language to test its speed performance.

5.1 The model

This section presents how the simulated arm model was constructed and how the output from it was used to reproduce the tracking model from the original report[5].

5.1.1 Starting with ideal gyroscope data

The model was created using Simulink in MATLAB R2020a and a Simulink package called Simscape Multibody [20]. It represents an arm with two joints, one in the shoulder and one in the elbow. On the upper arm and the forearm an IMU is attached respectively. The IMU can be simulated using the MATLAB package called “Sensor Fusion and Tracking Toolbox”. The data produced from these IMUs during motion with the arm is used later on to calculate the position of the elbow in real time. The basic model can be seen in figure 5.1.

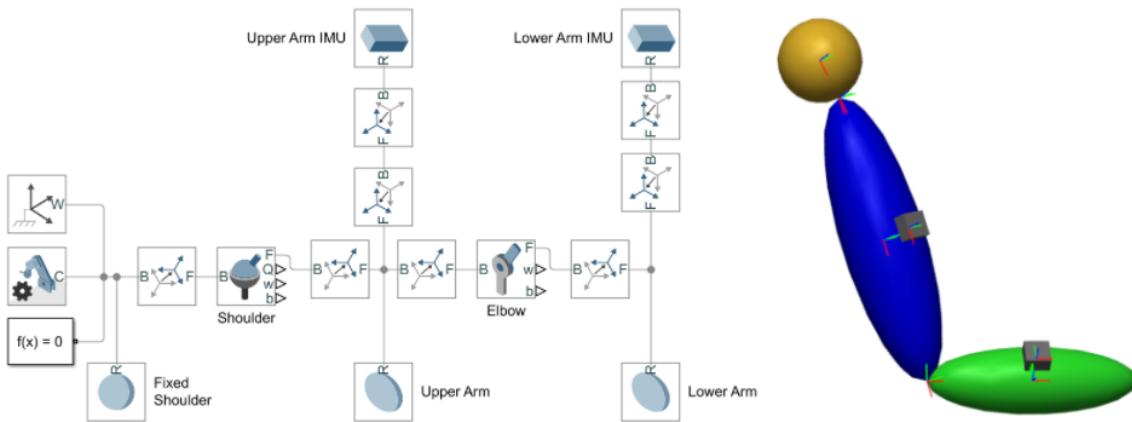


Figure 5.1: The basic Simulink arm model and the arm visualized. The arm was built from the shoulder represented by the yellow sphere connecting to the upper arm through a spherical joint and further to lower arm through a hinge joint. The grey cubes represent the IMUs connected to the upper and lower arm respectively.

The shoulder joint can be rotated in three degrees of freedom while the elbow is assumed to only rotate in two degrees of freedom. The elbow in the model can only rotate between 0 to 150 degrees to resemble normal motion range of a human body. When the elbow is fully straight, the extension is considered to be zero degrees. The upper arm has a length of 350 mm and the lower arm has a length of 250 mm.

To simulate motion, Simulink blocks were added to the model that enables external forces and torques to the upper arm and the lower arm. These are used as inputs to simulate different rotations of the arm. Several blocks were also added to output and visualize data during motion. The full model can be seen in 5.2.

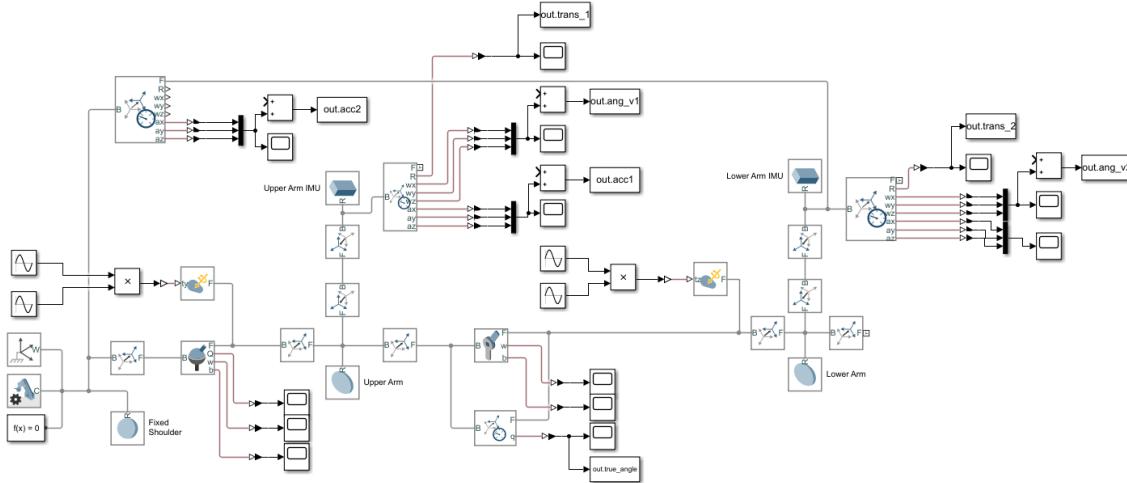


Figure 5.2: The Simulink arm model. Apart from the setup in Figure 5.1, this model also contains inputs to move the arm, outputs for the data needed for the algorithm and scopes to view the behavior of the model.

5.1.2 The Matlab script

The script first set up variables with values used by the Simulink model. Then, after the models simulation had been run, it stored output data from the model. The model outputs data used as input for the algorithm, the true angle of the joint and also transformation matrices used to transform the gravity into the local coordinate system for each IMU. The model simulation did not include any gravity so that was added to the acceleration of each IMU in the script.

The algorithm also requires the position of each IMU seen as an offset from the center of the joint. These were known values in the model but can be approximated using a calibration process described in [5]. The algorithms calculations were then done using the simulated sensor output and the result was plotted against the true angle from the model.

5.1.3 Simulated motion

One set of inputs to create a simple simulated motion was used throughout the process of creating the MATLAB implementation. After the method was implemented a wide range of motions were tested to assess how robust it is. The range of motion and the speed was varied and different arm orientations and shoulder rotations were explored. When weaknesses in the method were found, more specific motions were tested to explore and find the cause.

5.2 Adding noise to the IMUs

The IMUs used in the model are by default ideal IMUs. However, the properties of them can be adjusted using an IMU simulation model in MATLAB called imuSensor. The imuSensor has

properties to adjust the gyroscope sensor, called gyroparams, and adjust the accelerometer sensor, called accelparams. These classes make it easy to add different types of noise to the IMUs. The properties for adjusting the gyroscope or the accelerometer are divided into the following categories:

IMU model noise parameters

Parameter	Explanation	Unit
MeasurementRange	Maximum sensor reading	rad/s
Resolution	Resolution of sensor measurements	(rad/s)/LSB
ConstantBias	Constant sensor offset bias	rad/s
AxesMisalignment	Sensor axes skew	%
NoiseDensity	Power spectral density of sensor noise	(rad/s)/Hz
BiasInstability	Instability of the bias offset	(rad/s)
RandomWalk	Integrated white noise of sensor	(rad/s)(Hz)
TemperatureBias	Sensor bias from temperature	(rad/s)/°C
TemperatureScaleFactor	Scale factor error from temperature	%/°C
AccelerationBias	Sensor bias from linear acceleration	(rad/s)/(m/s ²)

ConstantBias and AxesMisalignment can be corrected through calibration before starting the VR-session. The temperature noise is unlikely to cause any major effects on the result since VR is played indoors where the temperature is relatively stable around 20°C. AccelerationBias is unlikely to happen for a long period. The remaining five IMU properties which will be tested in this report for the gyroscope and the accelerometer are listed below.

- MeasurementRange
- Resolution
- NoiseDensity
- BiasInstability
- RandomWalk

These parameters can be used to test the effect of adding noise to the IMUs. It is applied on the IMU data from the model. In other words, the data from the model is ideal, and the noise is added in the MATLAB script before the algorithm used to calculate the angle of the elbow. The noise levels are chosen based on a COTS IMU, more specifically the Murata scl3300 [19].

5.3 Evaluating latency

Examining the process of using the evaluated method, three factors contributing to the input lag can be found:

- The time it takes to transfer the sampled IMU data to the computer,
- The method uses a second order approximation of a derivative,
- The time it takes to calculate the angle.

5.3.1 Data transfer

As found by [21], the input lag for button presses using the HTC Vive Pro is on average 13.63 ms with a standard deviation of less than 5 ms. The buttons for this headset are on the hand controllers that use wireless communication. This would be the most practical way to transfer the sampled data from the IMUs and since the amount of data to transfer is also very small the resulting delay is assumed to be similar. The amount of data that has to be transferred each time the sensors are sampled is 48 bytes per IMU if 4-byte floating point precision is used.

5.3.2 Algorithmic delay

The second order approximation means that the calculation of the angle cannot take place until two later samples have been taken from the sensor. The contribution of this to the delay is therefore two times the sampling period.

5.3.3 Calculation time

The contribution to the delay caused by the time it takes to calculate the angle was evaluated by implementing the algorithm in a low-level language and testing its performance on VR capable hardware. The table below shows the minimum requirements for a PC to run the popular VR system called HTC Vive [22].

System requirements

Component	Recommended system requirements	Minimum system requirements
Processor	Intel Core i5-4590/AMD FX 8350 equivalent or better	Intel Core i5-4590/AMD FX 8350 equivalent or better
GPU	NVIDIA GeForce GTX 1060, AMD Radeon RX 480 equivalent or better	NVIDIA GeForce GTX 970, AMD Radeon R9 290 equivalent or better
Memory	4 GB RAM or more	4 GB RAM or more

The implementation was done in C++ and ran on a computer with the specifications shown in the table below.

Test system

Component	System
Processor	Intel Core i7-2600 (overclocked to 4 GHz)
GPU	NVIDIA GeForce GTX 1080
Memory	16 GB RAM

The implementation was verified by feeding it an input that had also been run through the MATLAB implementation confirming that it gave the same output. In order to evaluate the performance the code was compiled into an executable that ran ten million cycles. Each cycle's calculation time was saved and all times were then sorted and written to a file for analysis and plotting.

Chapter 6

Results

This chapter presents the results.

6.1 Verify the model

The first step was to verify that the algorithm implementation using MATLAB, based on data from the Simulink model, performs as expected. Figure 6.1 shows the difference between the true angle of the elbow, retrieved directly from the Simulink model, and the calculated angle of the elbow based of the two IMU:s attached to the upper and lower arm respectively. It also shows the estimated angle calculated based on solely the accelerometer as input data and solely using the gyroscope as input data. The arm started from a standstill position where the upper arm was pointing downwards and with the lower arm perpendicular to the upper arm. The simulation lasted for ten seconds while forces were added to the upper and lower arm to create a motion similar to Figure 6 in [5]. As seen in Figure 6.1, the estimated elbow angle is almost identical to the true angle. The root mean square error (RMSE) during the simulation was 0.29 degrees. The starting position and maximum bend of the arm during the movement can be seen in Figure 6.2.

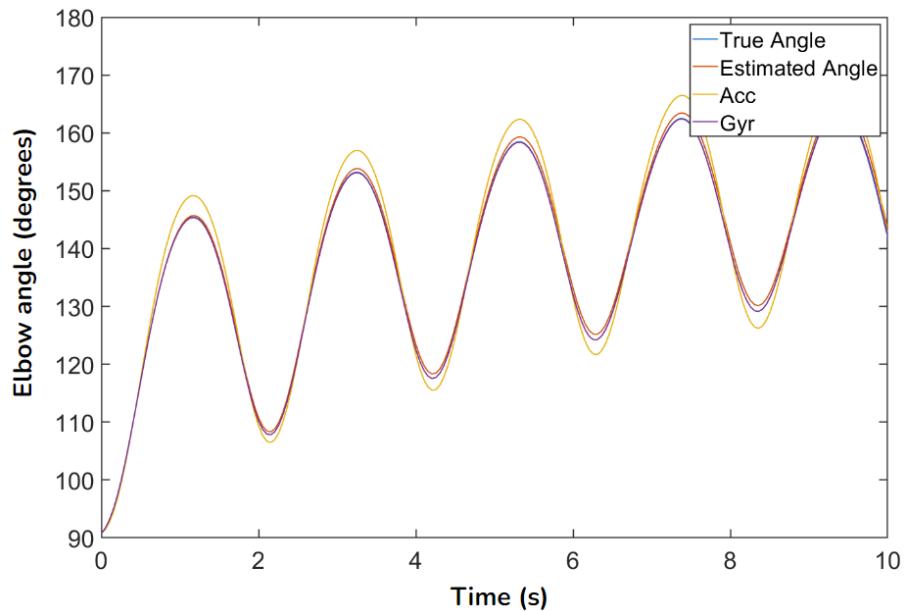


Figure 6.1: True angle (blue line) versus estimated angle (red line) without IMU noise or data delays. The graph also shows the estimated angle solely calculated using the accelerometer (yellow line) or by solely using the gyro meter (purple line).

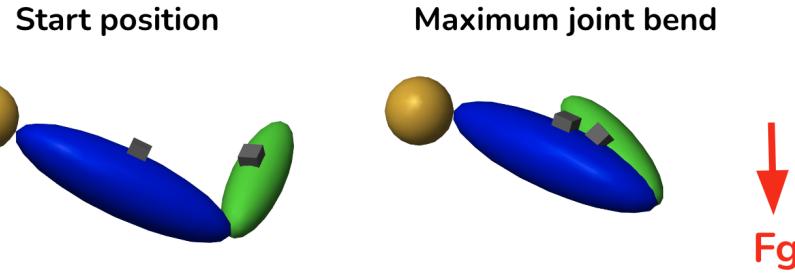


Figure 6.2: Start position and maximum joint bend of the arm for Figure 6.1.

6.2 Simulation of noise

Based on section 5.2, five different noise types for the gyroscope and the accelerometer were tested individually and combined. The noise levels used in the simulation was set based on the datasheet from an IMU model called Murata scl3300 [19]. The different noise types were added to the gyroscope and the accelerometer respectively. Figure 6.3 and Figure 6.4 shows the difference on each IMU with and without noise. Figure 6.5 shows the difference of the estimated elbow angle when all noise parameters are added to the IMUs. The RMSE remained below 4 degrees with all noise added to the IMUs.

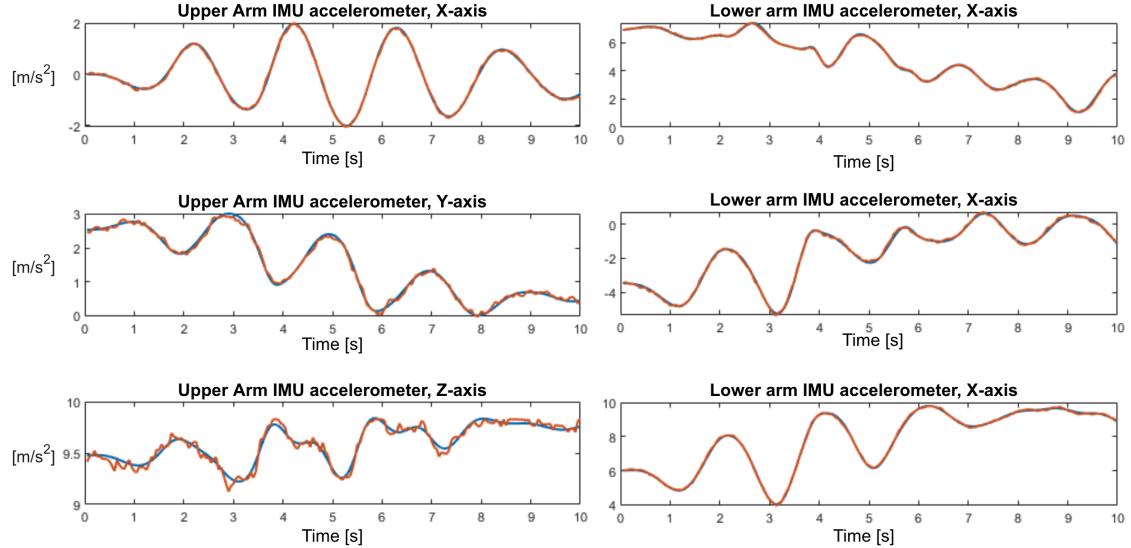


Figure 6.3: These plots show the difference between an ideal accelerometer (blue lines) and an accelerometer with noise (red lines). The left plots shows data from the IMU mounted on the upper arm and the plots on the right shows data from the IMU mounted on the lower arm. The top plots show the x-axes of the accelerometer, the middle plots show the y-axes and the bottom plots show the z-axes. The x-axes for all plots are in unit seconds and the y-axes for all plots is in unit acceleration m/s^2 .

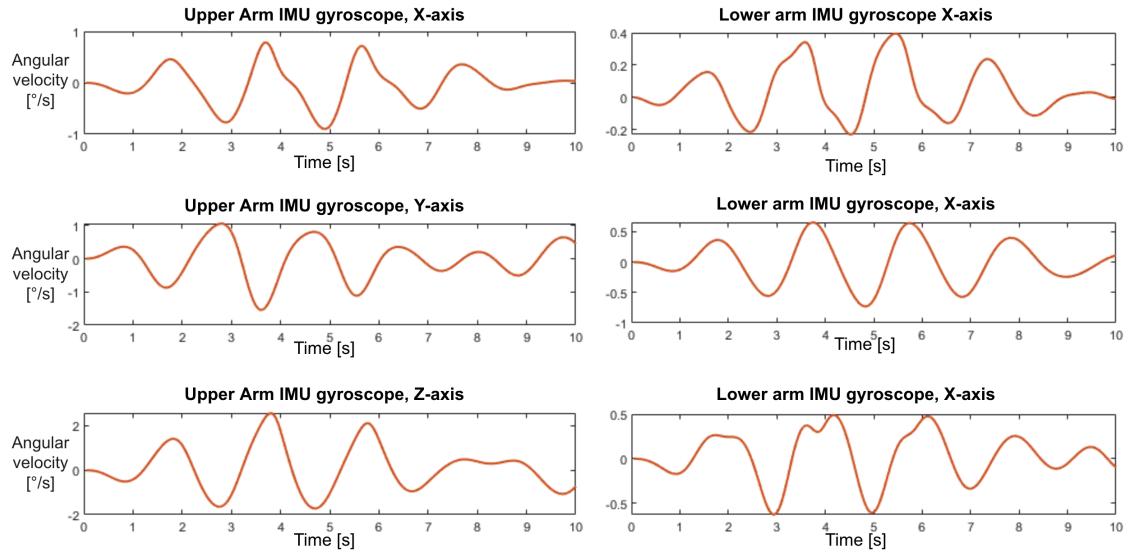


Figure 6.4: These plots show the difference between an ideal gyroscope (blue lines) and a gyroscope with noise (red lines). The plots are configured in the same way as the ones showed in Figure 6.3 expect for the y-axes which uses the unit angular velocity (rad/s). The noise had no visual effect on the plots which is why only the red lines can be seen.

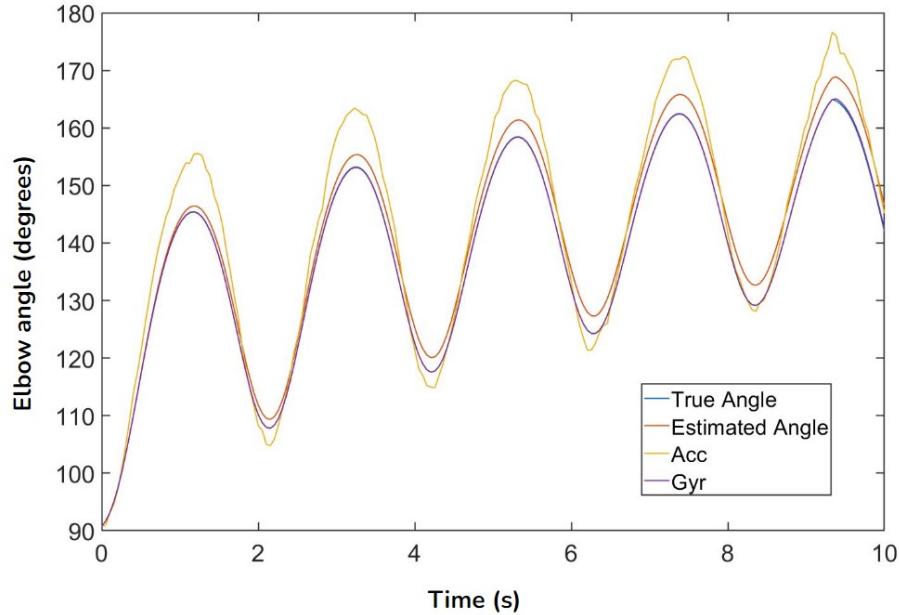


Figure 6.5: Angle estimation with IMU noise, combining both gyroscope and accelerometer estimations using a complementary filter.

6.3 Resulting input lag

In this section the input lag resulting from the contributing factors found in section 5.3 are presented.

6.3.1 Data transfer delay

As found by [21], delays for the HTC Vive Pro range from around 13 ms for button press input to around 65 ms for auditory output, in common applications. The transfer speed for the sensor data is assumed to lie between these numbers. The amount of data transferred if the sensor output is transferred as 4-byte floating point numbers is 24 bytes per sample. This adds up to a total transfer rate of 19.2 KB/s with four sensors (two for each elbow), if sampled at 200 Hz. For the Vive headset audio this number is 176.4 KB/s or 192 KB/s depending on if the headset is set to CD or DVD quality audio.

6.3.2 Numerical derivation delay

The numerical derivation used in the evaluated method requires a delay of two samples giving a delay that is linearly proportional to the sample rate. The delay at a few different sample rates can be seen in the table below.

50 Hz	70 Hz	100 Hz	200 Hz
40 ms	28.6 ms	20 ms	10 ms

6.3.3 Calculation delay

The delay between getting a signal from the IMU and having a calculated angle is dependent on two things, the calculation time and the frequency at which the IMU is sampled. The frequency matters as the algorithm used implements a second order numerical approximation of the derivative and it requires two more earlier sets of IMU data. At a sample frequency of 100 Hz this gives a delay of 20 ms and is the dominant source of delay. The computation time in contrast, when it was run on the testing hardware seen in section 3.1, was an average of 0.213 μ s. A more detailed breakdown of the performance test can be seen in the table below.

Calculation delay breakdown	
Number of samples	10,000,000
Total calculation time	2,427,723.4 μ s
Fastest calculation time	0.2 μ s
Slowest calculation time	245 μ s
Mean calculation time	0.2428 μ s
Standard deviation	0.2444 μ s
99.91% of times were faster than 1 μ s	

The reason that the calculation times can vary this much is that it was running on a computer. The operating system of that computer, in this case Windows, splits the resources between all running processes. Depending on that prioritization any given task will take a varying amount of time to complete. The goal of evaluating the methods viability for use in a consumer grade VR system ruled out other implementations that could give a constant calculation time. The distribution of the calculation times can also be seen represented in Figure 6.6 below.

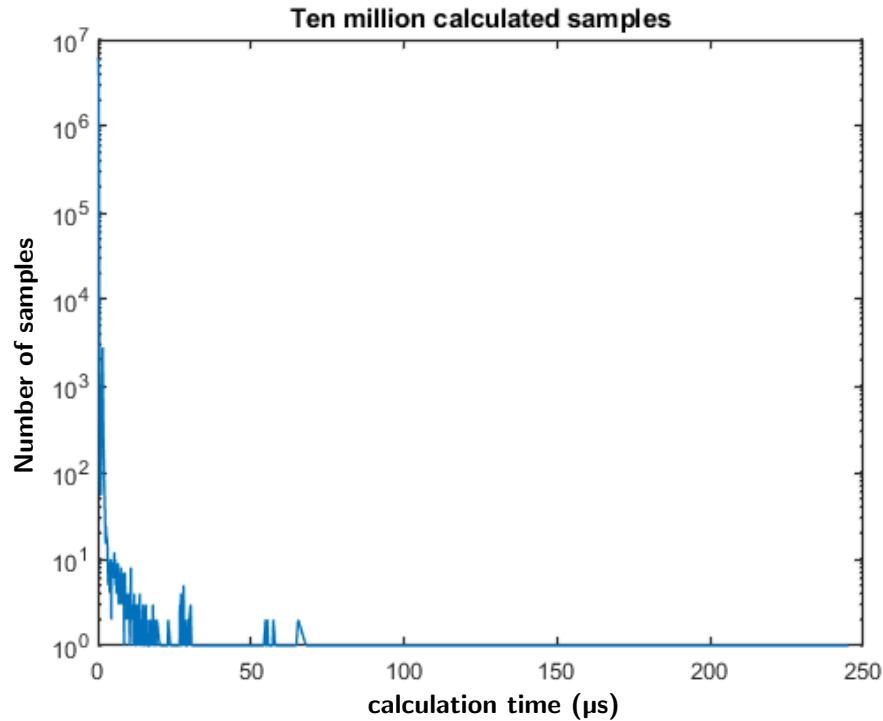


Figure 6.6: Graph over the calculation times with a log scale y-axis for the number of samples.

6.4 Adding both noise and lag to the model

Figure 6.7 shows the estimated angle when both IMU noise and data delays are added to the model. The RMSE was 4.3 with a low sample rate set to 20 hertz.

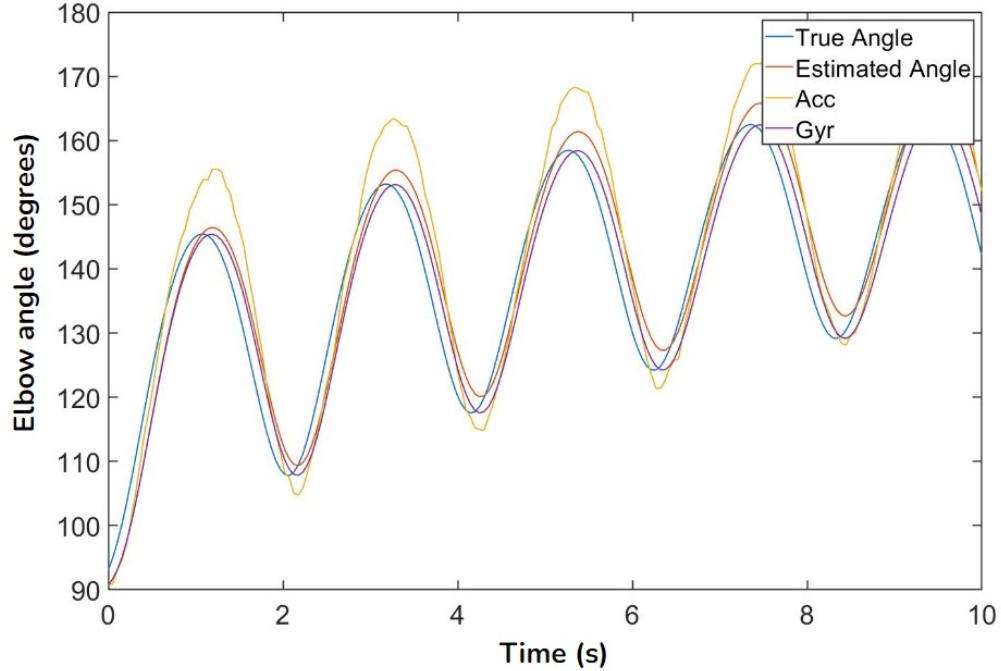


Figure 6.7: True angle versus estimated angle with IMU noise and data delays and using a sample rate of 20 hertz.

6.5 Testing different arm movements

This section presents the results from testing different movement patterns and orientations of the elbow.

6.5.1 Movement patterns

Several different movements of the arm were tested to verify the stability of the algorithm. Figure 6.8 varies both the motion speed and the length of the arm motion during 60 seconds. The RMSE was 0.45 degrees in this case.

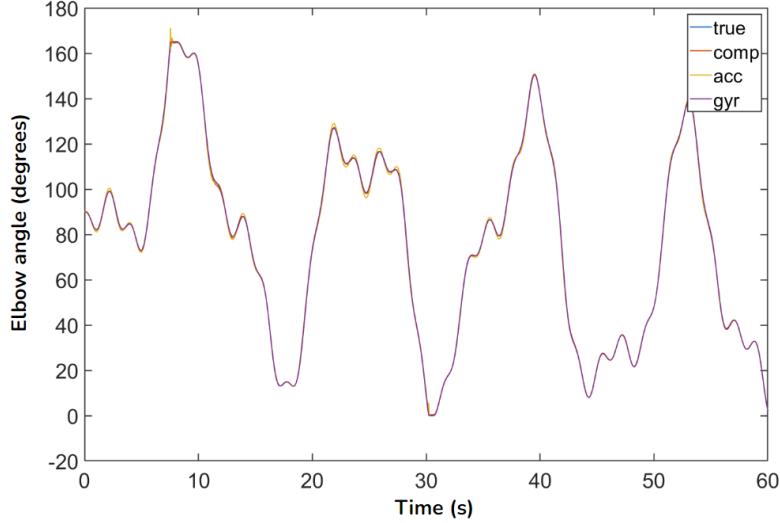


Figure 6.8: Random arm motion

Figure 6.9 shows a case when the arm moves fast in small, repetitive movements. The RMSE during 10 seconds is 2.75. The estimated arm angle is gradually drifting in this case and it can be seen that the angle estimation from the accelerometer is over-shooting when the arm is changing direction of the movement.

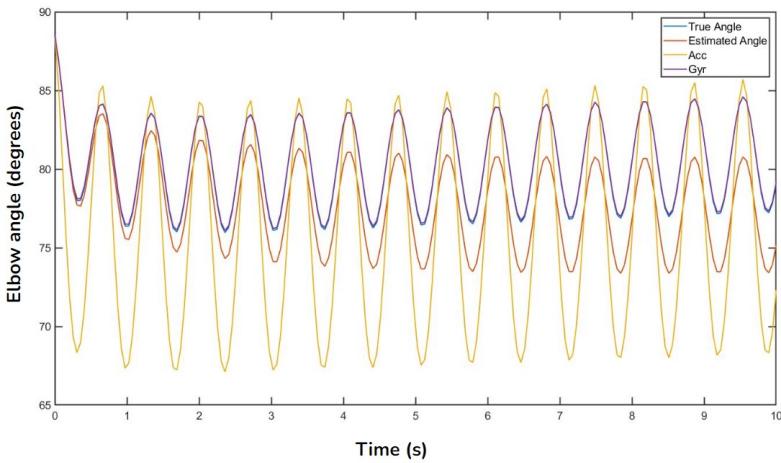


Figure 6.9: Repetitive, fast and small arm movement.

6.5.2 Movements in relation to gravity

This section explores the influence gravity has for the performance of the algorithm. The following six figures, Figure 6.10-6.15 shows the same arm movement and with the same starting angle of 90 degrees between the upper arm and the lower arm. The figures are divided into three different starting positions of the upper arm in relation to the gravitational force. The upper arm is facing downwards in the first two figures, Figure 6.10 and 6.11. Figure 6.10 is with normal gravity and Figure 6.11 shows the result when the gravity is ten times stronger than normal. The gravity was increased for the three test cases to show its importance for the algorithm's overall performance. It is mentioned in the original report [5] that the acceleration-based angle estimation is based on the assumption that the measured acceleration is dominated by gravity. As seen in the figures, the algorithm performs better when the gravity is a more dominant part of the overall acceleration.

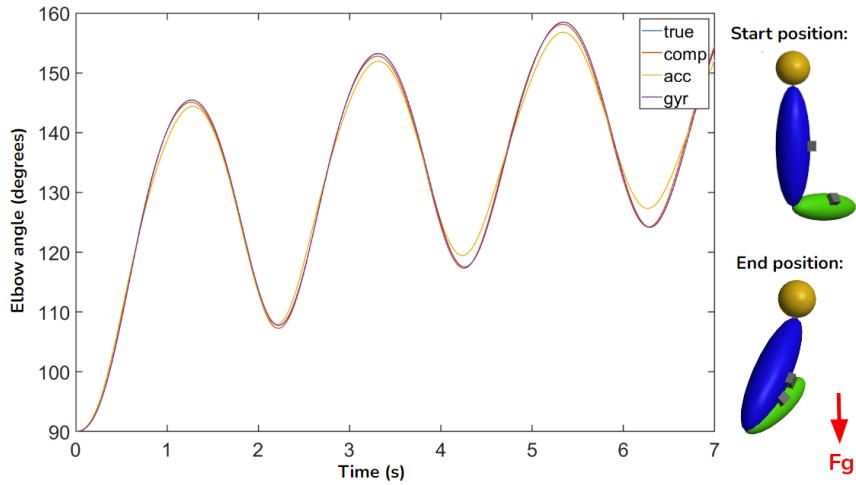


Figure 6.10: Normal gravity - upper arm facing downwards.

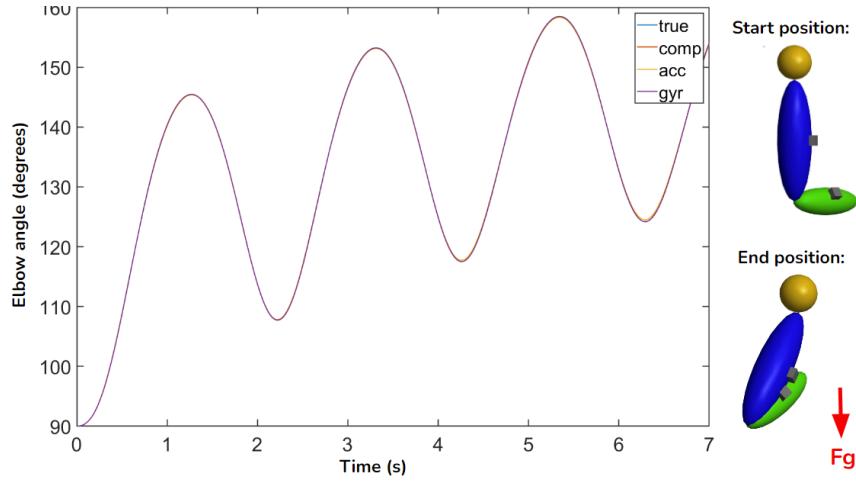


Figure 6.11: High gravity - upper arm facing downwards.

The upper arm starts in a perpendicular position (facing outwards from the shoulder) compared to the gravitational force in Figure 6.12 and 6.13. The lower arm is elevated, pointing 45 degrees upwards, putting the joint axes at an initial 45 degrees offset from gravity. It stays at an offset above 40 degrees throughout the recorded movement. Again, in the second figure 6.13 gravity which is ten times stronger compared to the normal gravity in Figure 6.12. While increased gravity still gives better performance, it is notable that the orientation of the elbow has a significant effect on the performance of the acceleration-based angle estimation.

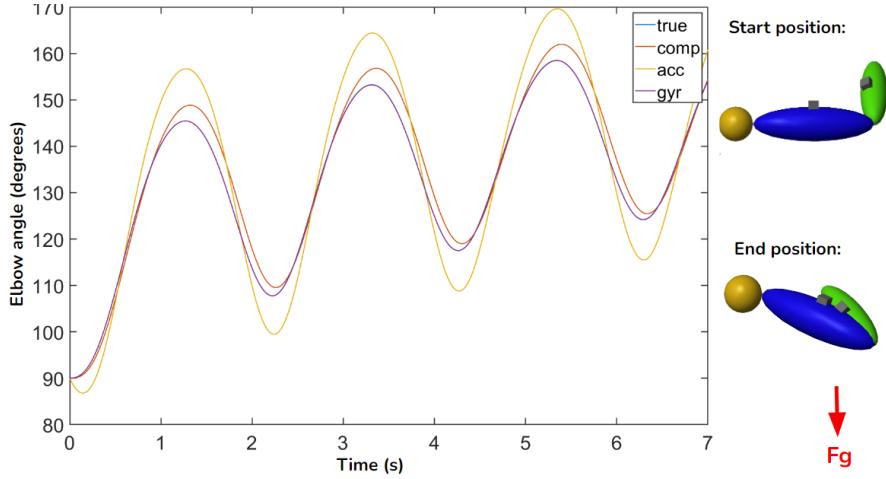


Figure 6.12: Normal gravity - upper arm perpendicular to body. In this case the RMSE of the acceleration-based angle estimation was above 8 degrees.

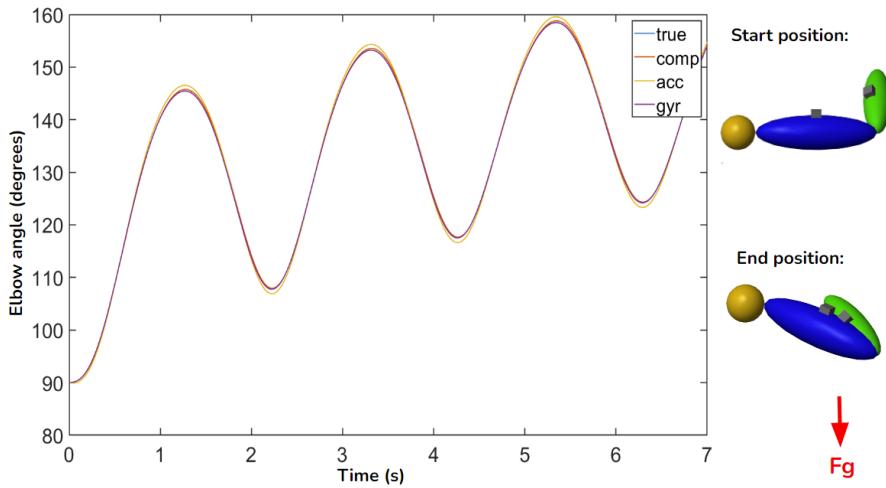


Figure 6.13: High gravity - upper arm perpendicular to body.

In the last two figures, Figure 6.14 and 6.15, both the upper arm and the lower arm is perpendicular to the gravitational force, making the joint axes collinear with gravity. The original report [5] mentions that getting closer to collinearity will make the algorithm more sensitive to measurement errors. However, these graphs were produced using ideal simulated values and still show that very significant errors are produced.

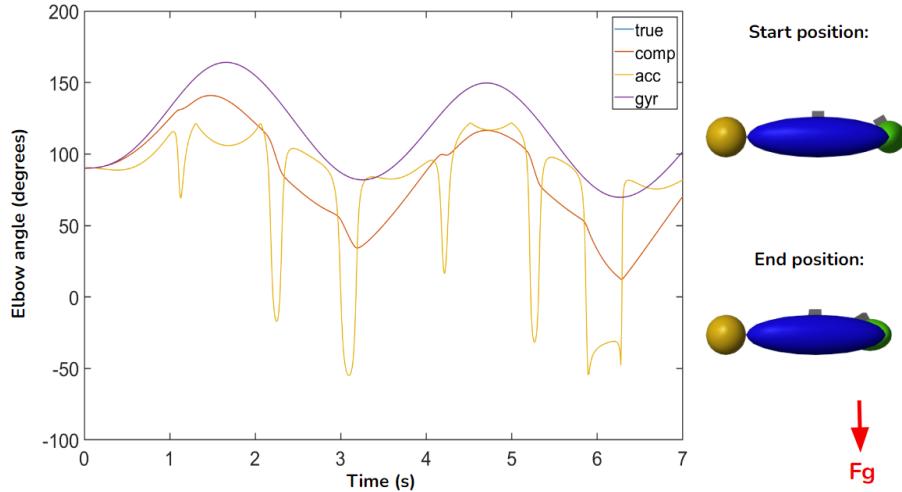


Figure 6.14: Normal gravity - joint axes collinear to gravity.

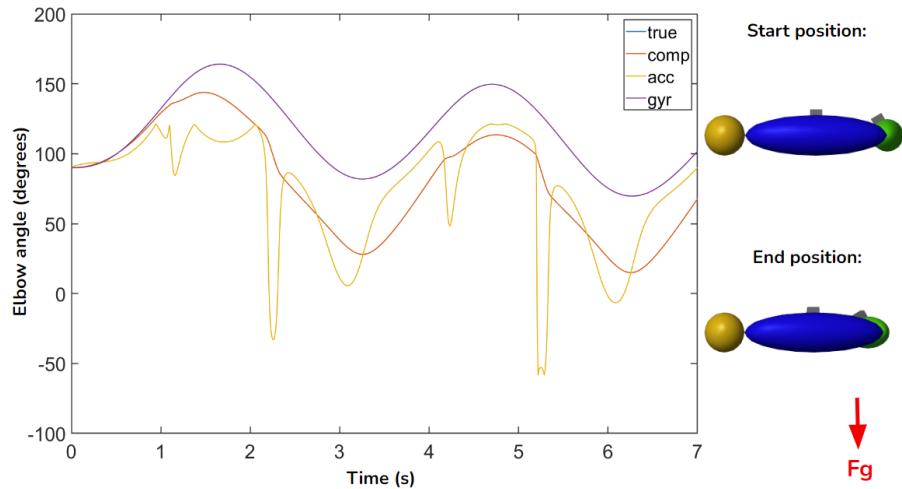


Figure 6.15: High gravity - joint axes collinear to gravity.

Chapter 7

Conclusions and future work

In this chapter the conclusions for the thesis is presented and suggestions for future work is discussed.

7.1 Conclusions

The method described in [5] could be reproduced and the RSME of the angle estimation was found to be similar or better than presented in the report in similar movement patterns. It was found to be resilient to the influence of noise, the RMSE staying below 4 degrees when noise, based on the datasheet of the Murata SCL3300 [19] IMU, is applied. The Murata SCL3300 was chosen as it is a well documented MEMS COTS IMU that specifies magnitudes of the different types of noise that were required for the evaluation.

The method does perform well in real-time, the largest contributor to latency being that the numerical derivation used requires two extra samples. As such the delay caused by the numerical derivation is proportional to the frequency at which the IMUs are sampled. According to [23] a latency of 50 ms for the HMD feels responsive. The tracking of the HMD is the most sensitive as any mismatch there is prone to cause cybersickness. The delay for the tracking of extremities is much less sensitive and can be as much as double that before causing significant loss of embodiment [15]. With this in mind a sample rate as low as 25 Hz would still leave 20 ms margin for the transfer of the data from the sensors to the computer. This is a sample rate that the IMU [19] examined in section 6.2 is more than capable of. Many MEMS COTS IMUs are however capable of sampling at much higher rates and for a commercial application it would likely come down to finding a good trade off between price, sampling rate and signal to noise ratio.

The downfall of the method was found to be that the angle estimation performance deteriorates significantly in cases where the acceleration measured by the IMUs gets closer to being collinear with the joint axes. As the movement of the elbow is much less constrained than the knee, common movement can align the joint axes with gravity making the acceleration-based estimation practically unusable. The original report [5] only states that the equations become more sensitive to *measurement error* when the acceleration vectors used are *almost collinear with the joint axes*. However, our simulations showed that even with ideal simulated values the angle estimation based on acceleration produces a significant error with these vectors being more than 40 degrees away from being collinear throughout the entire motion sequence. Above 8 degrees RMSE in the motion graphed in 6.12. As such this method is not found to be a good choice for tracking the angle of an elbow.

7.2 Future work

It is concluded that the method is not likely to work for this case. However, this study also showed that the method can perform well as long as the rotational axes of the tracked joint is close to being perpendicular with the gravitational force. Therefore, what follows is some suggestions for future work.

The first suggestion is to investigate other use cases for the algorithm based on the results in this report. For instance, a use case as in [5] where the motion is more limited compare to implementing it as in this report where the arms are free to move in any direction.

The IMUs evaluated in this thesis are COTS but with low noise and a relatively high price tag. However, the results indicate that the method can tolerate much higher noise levels. This would therefore make it interesting to test several IMUs with different price tags.

The impact of interference, and how it could be shielded against, would also be interesting to evaluate with a physical implementation.

If needed, a couple potential improvements can be examined. One thing that could be examined is a first-order approximation derivative of the angular velocity. This would reduce the derivative delay but as a tradeoff it loses precision in the approximation.

Another idea is to use other methods for sensor fusion. The method evaluated in this thesis [5] takes advantage of the natural constraints of hinge joints which limits its application to only tracking of specific joints. This is an actively researched area and many other methods that are not limited to hinge joints are available. In particular new methods of IMU orientation estimation, based on deep learning, like RIANN [16] and the methods presented in [17] look very promising. In the application of motion tracking for VR they look to be both fast enough on computer hardware and very robust. In addition, they work without an initial rest period and or predefined starting orientation making it easier to achieve a plug and play experience.

References

- [1] A. Ahmadi, E. Mitchell, C. Richter, F. Destelle, M. Gowing, N. E. O'Connor, and K. Moran, "Toward automatic activity classification and movement assessment during a sports training session," *IEEE Internet of Things Journal*, vol. 2, no. 1, pp. 23–32, 2015. doi: 10.1109/JIOT.2014.2377238
- [2] H. Zhou and H. Hu, "Human motion tracking for rehabilitation—a survey," *Biomedical signal processing and control*, vol. 3, no. 1, pp. 1–18, 2008.
- [3] H. Haggag, M. Hossny, S. Nahavandi, and D. Creighton, "Real time ergonomic assessment for assembly operations using kinect," in *2013 UKSim 15th International Conference on Computer Modelling and Simulation*. IEEE, 2013, pp. 495–500.
- [4] G. Welch and E. Foxlin, "Motion tracking: no silver bullet, but a respectable arsenal," *IEEE Computer Graphics and Applications*, vol. 22, no. 6, pp. 24–38, 2002. doi: 10.1109/MCG.2002.1046626
- [5] T. Seel, J. Raisch, and T. Schauer, "Imu-based joint angle measurement for gait analysis," *Sensors*, vol. 14, no. 4, pp. 6891–6909, 2014. doi: 10.3390/s140406891. [Online]. Available: <https://www.mdpi.com/1424-8220/14/4/6891>
- [6] Oculus quest. [Online]. Available: <https://www.meta.com/se/en/quest/products/quest-2/>
- [7] Htc vive. [Online]. Available: <https://www.vive.com/us/product/vive-pro2-full-kit/overview/>
- [8] P. Caserman, "Full-body motion tracking in immersive virtual reality-full-body motion reconstruction and recognition for immersive multiplayer serious games," 2021.
- [9] Vicon motion tracking. [Online]. Available: <https://www.vicon.com/hardware/cameras/>
- [10] Rokoko smartsuit pro. [Online]. Available: <https://www.rokoko.com/products/smartsuit-pro>
- [11] Xsens mvn link. [Online]. Available: <https://www.movella.com/products/motion-capture/xsens-mvn-link>
- [12] M. Caruso, A. M. Sabatini, D. Laidig, T. Seel, M. Knaflitz, U. Della Croce, and A. Cereatti, "Analysis of the accuracy of ten algorithms for orientation estimation using inertial and magnetic sensing under optimal conditions: One size does not fit all," *Sensors*, vol. 21, no. 7, p. 2543, 2021.
- [13] J. K. Lee and T. H. Jeon, "Imu-based but magnetometer-free joint angle estimation of constrained links," pp. 1–4, 2018. doi: 10.1109/ICSENS.2018.8589825. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8589825/references#references>
- [14] O. G. R. Wittmann, F.; Lambery, "Magnetometer-based drift correction during rest in imu arm motion tracking," *Sensors 2019*, 19, 1312., 2019.
- [15] P. Caserman, M. Martinussen, and S. Göbel, "Effects of end-to-end latency on user experience and performance in immersive virtual reality applications," in *Entertainment Computing and Serious Games*, E. van der Spek, S. Göbel, E. Y.-L. Do, E. Clua, and J. Baalsrud Hauge, Eds. Cham: Springer International Publishing, 2019. ISBN 978-3-030-34644-7 pp. 57–69.

- [16] D. Weber, C. Gühmann, and T. Seel, “Riann—a robust neural network outperforms attitude estimation filters,” *Ai*, vol. 2, no. 3, pp. 444–463, 2021.
- [17] A. A. Golroudbari and M. H. Sabour, “End-to-end deep learning framework for real-time inertial attitude estimation using 6dof imu,” *arXiv preprint arXiv:2302.06037*, 2023.
- [18] *Case study research in software engineering guidelines and examples*, 1st ed. Hoboken, N.J: Wiley, 2012. ISBN 1-280-58877-2
- [19] Imu scl3300-d01 from murata. [Online]. Available: https://www.murata.com/-/media/webrenewal/products/sensor/pdf/datasheet/datasheet_scl3300-d01.ashx?la=en&cvid=20210316063715000000
- [20] Simscape multibody - model and simulate multibody mechanical systems. [Online]. Available: <https://se.mathworks.com/products/simmechanics.html>
- [21] M. Le Chénéchal and J. Chatel-Goldman, “Htc vive pro time performance benchmark for scientific research,” in *Icat-Egve 2018*, 2018.
- [22] System requirements for htc vive. [Online]. Available: https://www.vive.com/us/support/vive/category_howto/what-are-the-system-requirements.html
- [23] K. Raaen and I. Kjellmo, “Measuring latency in virtual reality systems,” in *Entertainment Computing - ICEC 2015*, K. Chorianopoulos, M. Divitini, J. Baalsrud Hauge, L. Jaccheri, and R. Malaka, Eds. Cham: Springer International Publishing, 2015. ISBN 978-3-319-24589-8 pp. 457–462.

Appendix A

C++ Implementation

A.1 Vec3.h

```
#include <cstddef>
#pragma once

class Vec3 {
public:
    Vec3();
    Vec3(double, double, double);
    ~Vec3();

    Vec3 &operator=(const Vec3 &);

    Vec3 operator+(const Vec3 &) const;
    Vec3 &operator+=(const Vec3 &);

    Vec3 operator-(const Vec3 &) const;
    Vec3 &operator-=(const Vec3 &);

    Vec3 operator*(const double) const;
    Vec3 &operator*=(const double);

    Vec3 operator/(const double) const;
    Vec3 &operator/=(const double);

    double &operator[](std::size_t idx) { return mVec[idx]; }
    const double &operator[](std::size_t idx) const { return mVec[idx]; }

    void print() const;

private:
    double mVec[3];
};
```

A.2 Vec3.cpp

```
#include "Vec3.h"
#include <cmath>
#include <iostream>

Vec3::Vec3() {
    mVec[0] = 0.0;
    mVec[1] = 0.0;
    mVec[2] = 0.0;
}

Vec3::Vec3(double x, double y, double z) {
    mVec[0] = x;
    mVec[1] = y;
    mVec[2] = z;
}

Vec3::~Vec3() {}

Vec3 &Vec3::operator=(const Vec3 &inVec) {
    mVec[0] = inVec[0];
    mVec[1] = inVec[1];
    mVec[2] = inVec[2];

    return *this;
}

Vec3 Vec3::operator+(const Vec3 &inVec) const {
    return Vec3(mVec[0] + inVec[0], mVec[1] + inVec[1], mVec[2] + inVec[2]);
}

Vec3 &Vec3::operator+=(const Vec3 &inVec) {
    mVec[0] += inVec[0];
    mVec[1] += inVec[1];
    mVec[2] += inVec[2];
    return *this;
}

Vec3 Vec3::operator-(const Vec3 &inVec) const {
    return Vec3(mVec[0] - inVec[0], mVec[1] - inVec[1], mVec[2] - inVec[2]);
}

Vec3 &Vec3::operator--(const Vec3 &inVec) {
    mVec[0] -= inVec[0];
    mVec[1] -= inVec[1];
    mVec[2] -= inVec[2];
    return *this;
}

Vec3 Vec3::operator*(const double factor) const {
    return Vec3(mVec[0] * factor, mVec[1] * factor, mVec[2] * factor);
}

Vec3 &Vec3::operator*=(const double factor) {
    mVec[0] *= factor;
    mVec[1] *= factor;
    mVec[2] *= factor;

    return *this;
}

Vec3 Vec3::operator/(const double factor) const {
    return Vec3(mVec[0] / factor, mVec[1] / factor, mVec[2] / factor);
}

Vec3 &Vec3::operator/=(const double dNum) {
```

```
mVec[0] /= dNum;
mVec[1] /= dNum;
mVec[2] /= dNum;

return *this;
}

void Vec3::print() const {
    std::cout << "[" << mVec[0] << ", " << mVec[1] << ", " << mVec[2] << "]"
    std::cout << std::endl;
}
```

A.3 Joint.h

```
#pragma once
#include "MockSensor.h"
#include "Vec3.h"

class Joint {
public:
    Joint(Vec3 J1, Vec3 J2, Vec3 O1, Vec3 O2, Vec3 gravity1, Vec3 gravity2, double initialAngle,
          double gyroOffset, double accelOffset, double kalmanLamda);
    ~Joint();

    double CalculateAngle(SensorData data1, SensorData data2);
    void resetIntegration(void);

private:
    struct DataCache {
        DataCache(int length);
        DataCache(int length, Vec3 initData);
        ~DataCache();

        void NewData(Vec3 inData);
        Vec3 &operator[](int idx);
        const Vec3 &operator[](int idx) const;

        int mLength;
        Vec3 *mData;
        int mCurrent;
    };

    const Vec3 mJ1;
    const Vec3 mJ2;
    const Vec3 mX1;
    const Vec3 mX2;
    const Vec3 mY1;
    const Vec3 mY2;
    const Vec3 mO1;
    const Vec3 mO2;

    DataCache mAngularVelocities1;
    DataCache mAngularVelocities2;
    DataCache mAccelerations1;
    DataCache mAccelerations2;
    double mIntegration;
    double mLastIntegrationValue;
    double mInitialAngle;
    double mGyroOffset;
    double mAccelOffset;
    double mFilterLambda;
    double mLastGyroAngle;
    double mLastFilterAngle;
};
```

A.4 Joint.cpp

```
#include "Joint.h"
#include "MathFunctions.h"

Joint::Joint(Vec3 J1, Vec3 J2, Vec3 O1, Vec3 O2, Vec3 gravity1, Vec3 gravity2, double initialAngle,
            double gyroOffset, double accelOffset, double filterLambda)
: mJ1(J1), mJ2(J2), mX1(Cross(mJ1, Vec3(1.0, 0.0, 0.0))), mX2(Cross(mJ2, Vec3(1.0, 0.0, 0.0))),
  mY1(Cross(mJ1, mX1)), mY2(Cross(mJ2, mX2)), mO1(O1), mO2(O2), mAngularVelocities1(5),
  mAngularVelocities2(5), mAccelerations1(3, gravity1), mAccelerations2(3, gravity2),
  mIntegration(0.0), mLastIntegrationValue(0.0), mInitialAngle(initialAngle),
  mGyroOffset(gyroOffset), mAccelOffset(accelOffset), mFilterLambda(filterLambda),
  mLastGyroAngle(initialAngle), mLastFilterAngle(initialAngle) {}

Joint::~Joint() {}

double Joint::CalculateAngle(SensorData data1, SensorData data2) {
    mAngularVelocities1.NewData(data1.AngularVelocities);
    mAngularVelocities2.NewData(data2.AngularVelocities);
    mAccelerations1.NewData(data1.Accelerations);
    mAccelerations2.NewData(data2.Accelerations);

    // index 0 is what we just got, 1 (same as -4) is our oldest data
    // we are calculating acceleration for 2 time steps back
    Vec3 angularAcceleration1 =
        DerivativeApproximation(mAngularVelocities1[1], mAngularVelocities1[2],
                               mAngularVelocities1[4], mAngularVelocities1[0], data1.TimeStep);
    Vec3 angularAcceleration2 =
        DerivativeApproximation(mAngularVelocities2[1], mAngularVelocities2[2],
                               mAngularVelocities2[4], mAngularVelocities2[0], data2.TimeStep);

    double integrationValue = Dot(mAngularVelocities1[3], mJ1) - Dot(mAngularVelocities2[3], mJ2);

    // Gyroscope angle:
    mIntegration += TrapezoidalIntegration(mLastIntegrationValue, integrationValue, data1.TimeStep);
    double gyroAngle = mIntegration;

    mLastIntegrationValue = integrationValue;

    // Calculating acceleration due to rotation around the joint center
    Vec3 rotAcceleration1 = Cross(mAngularVelocities1[3], Cross(mAngularVelocities1[3], mO1)) +
        Cross(angularAcceleration1, mO1);
    Vec3 rotAcceleration2 = Cross(mAngularVelocities2[3], Cross(mAngularVelocities2[3], mO2)) +
        Cross(angularAcceleration2, mO2);

    Vec3 AccelApproximation1 = mAccelerations1[1] - rotAcceleration1;
    Vec3 AccelApproximation2 = mAccelerations2[1] - rotAcceleration2;

    double x1 = Dot(AccelApproximation1, mX1);
    double y1 = Dot(AccelApproximation1, mY1);
    double x2 = Dot(AccelApproximation2, mX2);
    double y2 = Dot(AccelApproximation2, mY2);

    double alpha_acc = abs(-acos((x1 * x2 + y1 * y2) / (norm2d(x1, y1) * norm2d(x2, y2))));

    double accelerationAngle = mAccelOffset + alpha_acc;
    gyroAngle += mGyroOffset;

    double filterAngle = mFilterLambda * accelerationAngle +
        (1.0 - mFilterLambda) * (mLastFilterAngle + gyroAngle - mLastGyroAngle);

    mLastFilterAngle = filterAngle;
    mLastGyroAngle = gyroAngle;

    return filterAngle;
}
```

```

void Joint::resetIntegration(void) {
    mIntegration = 0.0;
    mLastIntegrationValue = 0.0;
    mLastFilterAngle = mInitialAngle;
    mLastGyroAngle = mInitialAngle;
}

Joint::DataCache::DataCache(int length) : mLength(length), mCurrent(0) {

    mData = new Vec3[mLength];

    for (int i = 0; i < mLength; i++) {
        mData[i] = Vec3();
    }
}

Joint::DataCache::DataCache(int length, Vec3 initData) : mLength(length), mCurrent(0) {

    mData = new Vec3[mLength];

    for (int i = 0; i < mLength; i++) {
        mData[i] = initData;
    }
}

Joint::DataCache::~DataCache() { delete[] mData; }

void Joint::DataCache::NewData(Vec3 inData) {
    mCurrent = (mCurrent + 1) % mLength;
    mData[mCurrent] = inData;
}

Vec3 &Joint::DataCache::operator[](int idx) {
    int id = (mCurrent + idx) % mLength;
    if (id < 0) {
        id += mLength;
    }
    return mData[id];
}

const Vec3 &Joint::DataCache::operator[](int idx) const {
    int id = (mCurrent + idx) % mLength;
    if (id < 0) {
        id += mLength;
    }
    return mData[id];
}

```

A.5 MathFunctions.h

```
#pragma once
#include "Vec3.h"
#include <cmath>

static inline Vec3 DerivativeApproximation(const Vec3 &back2, const Vec3 &back1,
                                           const Vec3 &forward1, const Vec3 &forward2,
                                           const double deltaTime) {
    return Vec3((back2[0] - 8.0 * back1[0] + 8.0 * forward1[0] - forward2[0]) / (12 * deltaTime),
                (back2[1] - 8.0 * back1[1] + 8.0 * forward1[1] - forward2[1]) / (12 * deltaTime),
                (back2[2] - 8.0 * back1[2] + 8.0 * forward1[2] - forward2[2]) / (12 * deltaTime));
}

static inline Vec3 TrapezoidalIntegration(const Vec3 &a, const Vec3 &b, const double step) {
    return (a + b) * (step * 0.5);
}

static inline double TrapezoidalIntegration(const double &a, const double &b, const double step) {
    return (a + b) * (step * 0.5);
}

static inline double Dot(const Vec3 &a, const Vec3 &b) {
    return a[0] * b[0] + a[1] * b[1] + a[2] * b[2];
}

static inline Vec3 Cross(const Vec3 &a, const Vec3 &b) {
    return Vec3(a[1] * b[2] - a[2] * b[1], a[2] * b[0] - a[0] * b[2], a[0] * b[1] - a[1] * b[0]);
}

static inline double norm2d(double x, double y) { return sqrt(x * x + y * y); }
```

A.6 MockSensor.h

```
#pragma once
#include "Vec3.h"

struct SensorData {
    SensorData(Vec3 angularVelocities, Vec3 accelerations, double timeStep);

    Vec3 AngularVelocities;
    Vec3 Accelerations;
    double TimeStep;
};

class MockSensor {
public:
    MockSensor(const Vec3 angularVelocities[], const Vec3 accelerations[], const Vec3 gravity[],
               std::size_t length, double timeStep);
    ~MockSensor();

    SensorData getSensorData();

private:
    std::size_t mLength;
    const Vec3 *mAngularVelocities;
    const Vec3 *mAccelerations;
    const Vec3 *mGravity;
    double mTimeStep;

    std::size_t mCurrent;
};
```

A.7 MockSensor.cpp

```
#include "MockSensor.h"

MockSensor::MockSensor(const Vec3 angularVelocities[], const Vec3 accelerations[],
                      const Vec3 gravity[], std::size_t length, double timeStep)
: mAngularVelocities(angularVelocities), mAccelerations(accelerations), mGravity(gravity),
  mLength(length), mTimeStep(timeStep), mCurrent(0) {}

MockSensor::~MockSensor() {}

SensorData MockSensor::getSensorData() {
    if (mCurrent >= mLength) {
        mCurrent = 0;
    }

    SensorData data(mAngularVelocities[mCurrent], mAccelerations[mCurrent] + mGravity[mCurrent],
                    mTimeStep);
    mCurrent++;

    return data;
}

SensorData::SensorData(Vec3 angularVelocities, Vec3 accelerations, double timeStep)
: AngularVelocities(angularVelocities), Accelerations(accelerations), TimeStep(timeStep) {}
```

A.8 PerformanceTest.cpp

```
#include "Joint.h"
#include "MockSensor.h"
#include "TestData.h"
#include "Vec3.h"
#include <chrono>
#include <fstream>
#include <iostream>
#include <stdio.h>

int compare(const void *a, const void *b) { return (*(double *)a - *(double *)b); }

int main(int argc, char *argv[]) {
    Joint testJoint(Vec3(0.1422, -0.6649, 0.7333), Vec3(0.0063, -0.5054, 0.8629),
                    Vec3(0.175, 0.0, -0.05), Vec3(-0.125, 0.0, -0.04), Vec3(0.0, -2.539, -9.4757),
                    Vec3(-6.9367, 3.4684, -6.0074), 1.571356, 1.571356, -0.080112, 0.01);

    MockSensor sensor1(AngularVelocities1, Acceleration1, Gravity1, 201, 0.05);

    MockSensor sensor2(AngularVelocities2, Acceleration2, Gravity2, 201, 0.05);

    double timeTotal = 0.0;

    int samples = 10000000;

    double *times = new double[samples];

    for (int i = 0; i < samples; i++) {
        if (i == 5) {
            testJoint.resetIntegration();
        }

        std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
        double angle = testJoint.CalculateAngle(sensor1.getSensorData(), sensor2.getSensorData());
        std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
        times[i] = std::chrono::duration_cast<std::chrono::nanoseconds>(end - begin).count();
        timeTotal += times[i];
    }

    printf("Number of samples: %d\nTotal calculation time: %0.1f microseconds\nAverage: %0.3f "
          "microseconds\n\n",
          samples, timeTotal / 1000.0, timeTotal / (samples * 1000.0));

    std::cout << "Calculating more stats" << std::endl;

    std::cout << "Sorting times..." << std::endl;
    std::qsort(times, samples, sizeof(double), compare);

    int fivePercent = samples / 20;

    std::cout << "Quickest: " << times[0] / 1000.0 << " microseconds" << std::endl;
    std::cout << "5%: " << times[fivePercent] / 1000.0 << " microseconds" << std::endl;
    std::cout << "50%: " << times[samples / 2] / 1000.0 << " microseconds" << std::endl;
    std::cout << "95%: " << times[samples - fivePercent] / 1000.0 << " microseconds" << std::endl;
    std::cout << "Slowest 5: " << std::endl;
    for (size_t i = 6; i > 0; i--) {
        std::cout << times[samples - i] / 1000.0 << " microseconds" << std::endl;
    }

    std::cout << "How much is faster than 1 microsecond?" << std::endl;
    int i = 0;
    while (times[i] < 1000.0 && i < samples) {
        i++;
    }
    std::cout << i / double(samples) << "%" << std::endl << std::endl;
```

```
std::cout << "Writing times to file" << std::endl;
std::ofstream txtOut;
txtOut.open("times.csv");
for (size_t i = 0; i < samples; i++) {
    txtOut << times[i];
    if (i < samples - 1) {
        txtOut << "," << std::endl;
    }
}
txtOut.close();
std::cout << "Done" << std::endl;

delete[] times;

std::string wait;
std::cin >> wait;

return 0;
}
```

A.9 TestData.h

```
#pragma once
#include "Vec3.h"

const Vec3 AngularVelocities1[] = {
    Vec3(0.0000, 0.0000, 0.0000),    Vec3(-0.0004, -0.0002, -0.0001),
    Vec3(-0.0022, -0.0013, -0.0008), Vec3(-0.0057, -0.0039, -0.0025),
    Vec3(-0.0106, -0.0084, -0.0056), Vec3(-0.0166, -0.0151, -0.0102),
    Vec3(-0.0233, -0.0239, -0.0167), Vec3(-0.0301, -0.0347, -0.0252),
    Vec3(-0.0365, -0.0473, -0.0356), Vec3(-0.0419, -0.0614, -0.0480),
    Vec3(-0.0460, -0.0763, -0.0622), Vec3(-0.0482, -0.0915, -0.0778),
    Vec3(-0.0481, -0.1063, -0.0945), Vec3(-0.0456, -0.1199, -0.1117),
    Vec3(-0.0405, -0.1316, -0.1285), Vec3(-0.0329, -0.1406, -0.1441),
    Vec3(-0.0229, -0.1463, -0.1573), Vec3(-0.0108, -0.1480, -0.1671),
    Vec3(0.0029, -0.1452, -0.1721), Vec3(0.0177, -0.1375, -0.1714),
    Vec3(0.0330, -0.1245, -0.1639), Vec3(0.0485, -0.1063, -0.1491),
    Vec3(0.0637, -0.0827, -0.1267), Vec3(0.0783, -0.0541, -0.0971),
    Vec3(0.0922, -0.0207, -0.0611), Vec3(0.1052, 0.0168, -0.0200),
    Vec3(0.1174, 0.0576, 0.0243), Vec3(0.1286, 0.1009, 0.0699),
    Vec3(0.1386, 0.1453, 0.1143), Vec3(0.1470, 0.1894, 0.1556),
    Vec3(0.1533, 0.2318, 0.1917), Vec3(0.1566, 0.2707, 0.2211),
    Vec3(0.1559, 0.3046, 0.2429), Vec3(0.1503, 0.3320, 0.2570),
    Vec3(0.1389, 0.3515, 0.2638), Vec3(0.1212, 0.3622, 0.2645),
    Vec3(0.0972, 0.3633, 0.2610), Vec3(0.0679, 0.3546, 0.2555),
    Vec3(0.0348, 0.3359, 0.2500), Vec3(-0.0001, 0.3076, 0.2462),
    Vec3(-0.0347, 0.2702, 0.2445), Vec3(-0.0674, 0.2245, 0.2444),
    Vec3(-0.0973, 0.1717, 0.2441), Vec3(-0.1242, 0.1130, 0.2409),
    Vec3(-0.1481, 0.0500, 0.2321), Vec3(-0.1693, -0.0159, 0.2149),
    Vec3(-0.1876, -0.0834, 0.1875), Vec3(-0.2022, -0.1511, 0.1487),
    Vec3(-0.2121, -0.2179, 0.0980), Vec3(-0.2158, -0.2824, 0.0358),
    Vec3(-0.2118, -0.3435, -0.0368), Vec3(-0.1986, -0.3997, -0.1179),
    Vec3(-0.1757, -0.4496, -0.2049), Vec3(-0.1428, -0.4918, -0.2944),
    Vec3(-0.1013, -0.5248, -0.3824), Vec3(-0.0530, -0.5472, -0.4642),
    Vec3(-0.0013, -0.5578, -0.5349), Vec3(0.0502, -0.5559, -0.5891),
    Vec3(0.0977, -0.5408, -0.6219), Vec3(0.1382, -0.5123, -0.6293),
    Vec3(0.1702, -0.4705, -0.6089), Vec3(0.1938, -0.4157, -0.5602),
    Vec3(0.2105, -0.3484, -0.4853), Vec3(0.2230, -0.2695, -0.3886),
    Vec3(0.2339, -0.1802, -0.2765), Vec3(0.2453, -0.0824, -0.1562),
    Vec3(0.2587, 0.0215, -0.0356), Vec3(0.2748, 0.1284, 0.0779),
    Vec3(0.2930, 0.2348, 0.1782), Vec3(0.3120, 0.3364, 0.2603),
    Vec3(0.3292, 0.4291, 0.3210), Vec3(0.3408, 0.5091, 0.3589),
    Vec3(0.3423, 0.5730, 0.3745), Vec3(0.3291, 0.6188, 0.3705),
    Vec3(0.2977, 0.6453, 0.3522), Vec3(0.2475, 0.6527, 0.3272),
    Vec3(0.1826, 0.6418, 0.3058), Vec3(0.1121, 0.6135, 0.2983),
    Vec3(0.0473, 0.5685, 0.3117), Vec3(-0.0038, 0.5084, 0.3453),
    Vec3(-0.0392, 0.4360, 0.3908), Vec3(-0.0626, 0.3552, 0.4361),
    Vec3(-0.0793, 0.2694, 0.4709), Vec3(-0.0940, 0.1807, 0.4895),
    Vec3(-0.1102, 0.0903, 0.4896), Vec3(-0.1295, -0.0007, 0.4713),
    Vec3(-0.1519, -0.0916, 0.4350), Vec3(-0.1762, -0.1813, 0.3817),
    Vec3(-0.1996, -0.2688, 0.3126), Vec3(-0.2187, -0.3531, 0.2293),
    Vec3(-0.2301, -0.4329, 0.1342), Vec3(-0.2304, -0.5069, 0.0303),
    Vec3(-0.2177, -0.5736, -0.0787), Vec3(-0.1910, -0.6310, -0.1887),
    Vec3(-0.1510, -0.6772, -0.2950), Vec3(-0.0997, -0.7101, -0.3928),
    Vec3(-0.0406, -0.7278, -0.4773), Vec3(0.0224, -0.7286, -0.5439),
    Vec3(0.0848, -0.7114, -0.5888), Vec3(0.1432, -0.6756, -0.6094),
    Vec3(0.1948, -0.6210, -0.6043), Vec3(0.2386, -0.5486, -0.5741),
    Vec3(0.2746, -0.4597, -0.5211), Vec3(0.3041, -0.3568, -0.4490),
    Vec3(0.3285, -0.2428, -0.3627), Vec3(0.3492, -0.1215, -0.2681),
    Vec3(0.3668, 0.0028, -0.1708), Vec3(0.3814, 0.1256, -0.0763),
    Vec3(0.3918, 0.2423, 0.0104), Vec3(0.3962, 0.3487, 0.0858),
    Vec3(0.3921, 0.4413, 0.1472), Vec3(0.3767, 0.5175, 0.1938),
    Vec3(0.3478, 0.5756, 0.2264), Vec3(0.3046, 0.6153, 0.2481),
    Vec3(0.2488, 0.6371, 0.2638), Vec3(0.1853, 0.6418, 0.2795),
    Vec3(0.1214, 0.6307, 0.3010), Vec3(0.0643, 0.6048, 0.3309),
    Vec3(0.0188, 0.5658, 0.3680), Vec3(-0.0140, 0.5156, 0.4071),
    Vec3(-0.0362, 0.4566, 0.4415), Vec3(-0.0512, 0.3905, 0.4658),
```

```

    Vec3(-0.0623, 0.3188, 0.4769), Vec3(-0.0722, 0.2426, 0.4737),
    Vec3(-0.0829, 0.1631, 0.4565), Vec3(-0.0952, 0.0812, 0.4262),
    Vec3(-0.1092, -0.0016, 0.3842), Vec3(-0.1242, -0.0841, 0.3321),
    Vec3(-0.1386, -0.1648, 0.2715), Vec3(-0.1508, -0.2423, 0.2044),
    Vec3(-0.1588, -0.3152, 0.1329), Vec3(-0.1610, -0.3819, 0.0593),
    Vec3(-0.1562, -0.4410, -0.0140), Vec3(-0.1438, -0.4909, -0.0845),
    Vec3(-0.1239, -0.5302, -0.1498), Vec3(-0.0971, -0.5576, -0.2076),
    Vec3(-0.0647, -0.5720, -0.2563), Vec3(-0.0282, -0.5727, -0.2943),
    Vec3(0.0107, -0.5594, -0.3207), Vec3(0.0503, -0.5324, -0.3352),
    Vec3(0.0889, -0.4924, -0.3379), Vec3(0.1252, -0.4407, -0.3294),
    Vec3(0.1579, -0.3791, -0.3107), Vec3(0.1859, -0.3097, -0.2829),
    Vec3(0.2083, -0.2349, -0.2476), Vec3(0.2247, -0.1569, -0.2063),
    Vec3(0.2344, -0.0783, -0.1607), Vec3(0.2376, -0.0013, -0.1122),
    Vec3(0.2341, 0.0723, -0.0624), Vec3(0.2245, 0.1407, -0.0128),
    Vec3(0.2094, 0.2027, 0.0354), Vec3(0.1896, 0.2571, 0.0809),
    Vec3(0.1661, 0.3031, 0.1227), Vec3(0.1400, 0.3404, 0.1600),
    Vec3(0.1121, 0.3686, 0.1918), Vec3(0.0836, 0.3876, 0.2177),
    Vec3(0.0551, 0.3974, 0.2371), Vec3(0.0275, 0.3984, 0.2497),
    Vec3(0.0014, 0.3909, 0.2554), Vec3(-0.0229, 0.3754, 0.2541),
    Vec3(-0.0448, 0.3525, 0.2462), Vec3(-0.0642, 0.3231, 0.2320),
    Vec3(-0.0806, 0.2880, 0.2121), Vec3(-0.0940, 0.2483, 0.1875),
    Vec3(-0.1040, 0.2050, 0.1589), Vec3(-0.1105, 0.1593, 0.1274),
    Vec3(-0.1136, 0.1124, 0.0943), Vec3(-0.1131, 0.0654, 0.0604),
    Vec3(-0.1094, 0.0195, 0.0272), Vec3(-0.1026, -0.0243, -0.0046),
    Vec3(-0.0932, -0.0651, -0.0337), Vec3(-0.0817, -0.1021, -0.0593),
    Vec3(-0.0688, -0.1350, -0.0806), Vec3(-0.0553, -0.1635, -0.0969),
    Vec3(-0.0418, -0.1874, -0.1077), Vec3(-0.0291, -0.2069, -0.1126),
    Vec3(-0.0176, -0.2220, -0.1116), Vec3(-0.0078, -0.2329, -0.1047),
    Vec3(0.0002, -0.2399, -0.0923), Vec3(0.0064, -0.2431, -0.0750),
    Vec3(0.0111, -0.2426, -0.0538), Vec3(0.0147, -0.2386, -0.0296),
    Vec3(0.0175, -0.2315, -0.0035), Vec3(0.0199, -0.2213, 0.0232),
    Vec3(0.0221, -0.2084, 0.0493), Vec3(0.0243, -0.1931, 0.0737),
    Vec3(0.0264, -0.1757, 0.0950), Vec3(0.0283, -0.1563, 0.1123),
    Vec3(0.0298, -0.1354, 0.1246), Vec3(0.0304, -0.1130, 0.1310),
    Vec3(0.0299, -0.0895, 0.1312), Vec3(0.0282, -0.0654, 0.1248),
    Vec3(0.0251, -0.0409, 0.1121), Vec3(0.0208, -0.0166, 0.0937),
    Vec3(0.0156, 0.0069, 0.0703), Vec3(0.0099, 0.0292, 0.0433),
    Vec3(0.0042, 0.0497, 0.0138), Vec3(-0.0010, 0.0681, -0.0165),
    Vec3(-0.0052, 0.0841, -0.0461), Vec3(-0.0080, 0.0975, -0.0736),
    Vec3(-0.0091, 0.1085, -0.0976),
};


```

```

const Vec3 Acceleration1[] = {Vec3(0.0000, 0.0000, -0.0000), Vec3(0.0000, 0.0003, -0.0005),
    Vec3(0.0000, 0.0011, -0.0018), Vec3(0.0000, 0.0023, -0.0035),
    Vec3(0.0000, 0.0038, -0.0056), Vec3(0.0000, 0.0055, -0.0077),
    Vec3(0.0001, 0.0074, -0.0099), Vec3(0.0002, 0.0094, -0.0118),
    Vec3(0.0003, 0.0113, -0.0135), Vec3(0.0005, 0.0131, -0.0147),
    Vec3(0.0008, 0.0148, -0.0153), Vec3(0.0012, 0.0160, -0.0153),
    Vec3(0.0016, 0.0168, -0.0145), Vec3(0.0020, 0.0169, -0.0129),
    Vec3(0.0025, 0.0162, -0.0105), Vec3(0.0028, 0.0146, -0.0073),
    Vec3(0.0030, 0.0119, -0.0034), Vec3(0.0028, 0.0081, 0.0012),
    Vec3(0.0023, 0.0031, 0.0062), Vec3(0.0014, -0.0030, 0.0116),
    Vec3(0.0001, -0.0098, 0.0171), Vec3(-0.0016, -0.0171, 0.0225),
    Vec3(-0.0034, -0.0245, 0.0278), Vec3(-0.0053, -0.0313, 0.0326),
    Vec3(-0.0069, -0.0372, 0.0369), Vec3(-0.0079, -0.0417, 0.0404),
    Vec3(-0.0081, -0.0443, 0.0430), Vec3(-0.0075, -0.0449, 0.0444),
    Vec3(-0.0059, -0.0434, 0.0445), Vec3(-0.0034, -0.0399, 0.0431),
    Vec3(-0.0004, -0.0346, 0.0401), Vec3(0.0027, -0.0281, 0.0354),
    Vec3(0.0057, -0.0207, 0.0291), Vec3(0.0081, -0.0131, 0.0215),
    Vec3(0.0097, -0.0060, 0.0130), Vec3(0.0103, 0.0000, 0.0038),
    Vec3(0.0100, 0.0044, -0.0056), Vec3(0.0087, 0.0068, -0.0149),
    Vec3(0.0069, 0.0073, -0.0239), Vec3(0.0046, 0.0065, -0.0324),
    Vec3(0.0020, 0.0053, -0.0402), Vec3(-0.0006, 0.0049, -0.0471),
    Vec3(-0.0031, 0.0063, -0.0528), Vec3(-0.0055, 0.0102, -0.0572),
    Vec3(-0.0078, 0.0164, -0.0604), Vec3(-0.0098, 0.0247, -0.0624),
    Vec3(-0.0114, 0.0343, -0.0635), Vec3(-0.0125, 0.0445, -0.0638),
};

```

$\text{Vec3}(-0.0127, 0.0546, -0.0635), \text{Vec3}(-0.0119, 0.0639, -0.0624),$
 $\text{Vec3}(-0.0097, 0.0721, -0.0604), \text{Vec3}(-0.0059, 0.0787, -0.0571),$
 $\text{Vec3}(-0.0005, 0.0830, -0.0519), \text{Vec3}(0.0063, 0.0846, -0.0444),$
 $\text{Vec3}(0.0140, 0.0827, -0.0341), \text{Vec3}(0.0217, 0.0764, -0.0211),$
 $\text{Vec3}(0.0283, 0.0649, -0.0058), \text{Vec3}(0.0323, 0.0477, 0.0109),$
 $\text{Vec3}(0.0327, 0.0251, 0.0280), \text{Vec3}(0.0285, -0.0017, 0.0441),$
 $\text{Vec3}(0.0195, -0.0303, 0.0585), \text{Vec3}(0.0066, -0.0580, 0.0708),$
 $\text{Vec3}(-0.0085, -0.0818, 0.0811), \text{Vec3}(-0.0235, -0.0997, 0.0897),$
 $\text{Vec3}(-0.0360, -0.1105, 0.0967), \text{Vec3}(-0.0440, -0.1140, 0.1022),$
 $\text{Vec3}(-0.0464, -0.1110, 0.1056), \text{Vec3}(-0.0429, -0.1025, 0.1064),$
 $\text{Vec3}(-0.0343, -0.0896, 0.1037), \text{Vec3}(-0.0218, -0.0736, 0.0969),$
 $\text{Vec3}(-0.0076, -0.0556, 0.0856), \text{Vec3}(0.0065, -0.0367, 0.0699),$
 $\text{Vec3}(0.0183, -0.0182, 0.0508), \text{Vec3}(0.0265, -0.0016, 0.0297),$
 $\text{Vec3}(0.0304, 0.0109, 0.0084), \text{Vec3}(0.0302, 0.0165, -0.0115),$
 $\text{Vec3}(0.0265, 0.0130, -0.0291), \text{Vec3}(0.0206, -0.0001, -0.0445),$
 $\text{Vec3}(0.0139, -0.0187, -0.0581), \text{Vec3}(0.0079, -0.0345, -0.0694),$
 $\text{Vec3}(0.0034, -0.0404, -0.0776), \text{Vec3}(0.0003, -0.0350, -0.0824),$
 $\text{Vec3}(-0.0022, -0.0215, -0.0847), \text{Vec3}(-0.0048, -0.0041, -0.0856),$
 $\text{Vec3}(-0.0079, 0.0143, -0.0854), \text{Vec3}(-0.0112, 0.0321, -0.0843),$
 $\text{Vec3}(-0.0146, 0.0487, -0.0825), \text{Vec3}(-0.0176, 0.0635, -0.0804),$
 $\text{Vec3}(-0.0196, 0.0761, -0.0782), \text{Vec3}(-0.0202, 0.0863, -0.0762),$
 $\text{Vec3}(-0.0188, 0.0938, -0.0743), \text{Vec3}(-0.0149, 0.0985, -0.0717),$
 $\text{Vec3}(-0.0084, 0.1005, -0.0676), \text{Vec3}(0.0003, 0.0995, -0.0608),$
 $\text{Vec3}(0.0106, 0.0954, -0.0504), \text{Vec3}(0.0212, 0.0877, -0.0360),$
 $\text{Vec3}(0.0306, 0.0760, -0.0174), \text{Vec3}(0.0371, 0.0600, 0.0043),$
 $\text{Vec3}(0.0393, 0.0400, 0.0278), \text{Vec3}(0.0362, 0.0169, 0.0515),$
 $\text{Vec3}(0.0278, -0.0077, 0.0735), \text{Vec3}(0.0150, -0.0317, 0.0925),$
 $\text{Vec3}(-0.0004, -0.0529, 0.1076), \text{Vec3}(-0.0162, -0.0698, 0.1185),$
 $\text{Vec3}(-0.0299, -0.0816, 0.1252), \text{Vec3}(-0.0396, -0.0882, 0.1277),$
 $\text{Vec3}(-0.0439, -0.0900, 0.1262), \text{Vec3}(-0.0427, -0.0878, 0.1206),$
 $\text{Vec3}(-0.0363, -0.0824, 0.1111), \text{Vec3}(-0.0260, -0.0744, 0.0978),$
 $\text{Vec3}(-0.0138, -0.0644, 0.0811), \text{Vec3}(-0.0014, -0.0534, 0.0620),$
 $\text{Vec3}(0.0093, -0.0425, 0.0417), \text{Vec3}(0.0171, -0.0332, 0.0217),$
 $\text{Vec3}(0.0215, -0.0274, 0.0032), \text{Vec3}(0.0227, -0.0262, -0.0130),$
 $\text{Vec3}(0.0215, -0.0294, -0.0269), \text{Vec3}(0.0190, -0.0342, -0.0387),$
 $\text{Vec3}(0.0162, -0.0368, -0.0486), \text{Vec3}(0.0136, -0.0342, -0.0567),$
 $\text{Vec3}(0.0110, -0.0262, -0.0632), \text{Vec3}(0.0080, -0.0142, -0.0684),$
 $\text{Vec3}(0.0044, -0.0002, -0.0726), \text{Vec3}(0.0002, 0.0140, -0.0757),$
 $\text{Vec3}(-0.0043, 0.0274, -0.0776), \text{Vec3}(-0.0088, 0.0394, -0.0784),$
 $\text{Vec3}(-0.0127, 0.0497, -0.0780), \text{Vec3}(-0.0156, 0.0580, -0.0766),$
 $\text{Vec3}(-0.0171, 0.0642, -0.0743), \text{Vec3}(-0.0171, 0.0681, -0.0713),$
 $\text{Vec3}(-0.0154, 0.0698, -0.0675), \text{Vec3}(-0.0120, 0.0693, -0.0627),$
 $\text{Vec3}(-0.0073, 0.0668, -0.0565), \text{Vec3}(-0.0016, 0.0626, -0.0485),$
 $\text{Vec3}(0.0045, 0.0567, -0.0386), \text{Vec3}(0.0101, 0.0495, -0.0268),$
 $\text{Vec3}(0.0148, 0.0411, -0.0131), \text{Vec3}(0.0177, 0.0319, 0.0018),$
 $\text{Vec3}(0.0186, 0.0221, 0.0173), \text{Vec3}(0.0174, 0.0120, 0.0326),$
 $\text{Vec3}(0.0142, 0.0019, 0.0467), \text{Vec3}(0.0094, -0.0079, 0.0589),$
 $\text{Vec3}(0.0038, -0.0171, 0.0687), \text{Vec3}(-0.0019, -0.0256, 0.0756),$
 $\text{Vec3}(-0.0071, -0.0331, 0.0797), \text{Vec3}(-0.0113, -0.0396, 0.0809),$
 $\text{Vec3}(-0.0141, -0.0448, 0.0796), \text{Vec3}(-0.0153, -0.0487, 0.0761),$
 $\text{Vec3}(-0.0149, -0.0512, 0.0708), \text{Vec3}(-0.0132, -0.0521, 0.0642),$
 $\text{Vec3}(-0.0103, -0.0514, 0.0565), \text{Vec3}(-0.0067, -0.0491, 0.0482),$
 $\text{Vec3}(-0.0027, -0.0454, 0.0396), \text{Vec3}(0.0012, -0.0403, 0.0307),$
 $\text{Vec3}(0.0048, -0.0341, 0.0218), \text{Vec3}(0.0078, -0.0269, 0.0130),$
 $\text{Vec3}(0.0099, -0.0191, 0.0045), \text{Vec3}(0.0110, -0.0109, -0.0038),$
 $\text{Vec3}(0.0111, -0.0028, -0.0116), \text{Vec3}(0.0102, 0.0051, -0.0189),$
 $\text{Vec3}(0.0085, 0.0125, -0.0255), \text{Vec3}(0.0062, 0.0190, -0.0313),$
 $\text{Vec3}(0.0035, 0.0246, -0.0363), \text{Vec3}(0.0007, 0.0289, -0.0404),$
 $\text{Vec3}(-0.0019, 0.0320, -0.0435), \text{Vec3}(-0.0042, 0.0338, -0.0455),$
 $\text{Vec3}(-0.0058, 0.0343, -0.0464), \text{Vec3}(-0.0068, 0.0337, -0.0462),$
 $\text{Vec3}(-0.0070, 0.0320, -0.0450), \text{Vec3}(-0.0066, 0.0294, -0.0428),$
 $\text{Vec3}(-0.0057, 0.0260, -0.0397), \text{Vec3}(-0.0044, 0.0219, -0.0360),$
 $\text{Vec3}(-0.0029, 0.0172, -0.0319), \text{Vec3}(-0.0013, 0.0121, -0.0274),$
 $\text{Vec3}(0.0002, 0.0066, -0.0228), \text{Vec3}(0.0015, 0.0010, -0.0182),$
 $\text{Vec3}(0.0026, -0.0047, -0.0138), \text{Vec3}(0.0033, -0.0101, -0.0095),$
 $\text{Vec3}(0.0037, -0.0151, -0.0054), \text{Vec3}(0.0039, -0.0194, -0.0016),$

```

    Vec3(0.0038, -0.0228, 0.0021),   Vec3(0.0036, -0.0251, 0.0055),
    Vec3(0.0033, -0.0263, 0.0087),   Vec3(0.0029, -0.0263, 0.0116),
    Vec3(0.0025, -0.0251, 0.0141),   Vec3(0.0021, -0.0226, 0.0163),
    Vec3(0.0016, -0.0190, 0.0183),   Vec3(0.0011, -0.0145, 0.0200),
    Vec3(0.0005, -0.0090, 0.0215),   Vec3(-0.0002, -0.0030, 0.0227),
    Vec3(-0.0010, 0.0032, 0.0237),   Vec3(-0.0019, 0.0095, 0.0243),
    Vec3(-0.0027, 0.0153, 0.0244),   Vec3(-0.0035, 0.0205, 0.0241),
    Vec3(-0.0040, 0.0247, 0.0232),   Vec3(-0.0043, 0.0277, 0.0218),
    Vec3(-0.0043, 0.0294, 0.0198),   Vec3(-0.0039, 0.0296, 0.0175),
    Vec3(-0.0032, 0.0283, 0.0149),   Vec3(-0.0024, 0.0257, 0.0123),
    Vec3(-0.0014, 0.0219, 0.0098)};
```

const Vec3 Gravity1[] = {Vec3(0.0000, -2.5390, -9.4757), Vec3(0.0000, -2.5391, -9.4757),
 Vec3(0.0003, -2.5396, -9.4756), Vec3(0.0012, -2.5414, -9.4751),
 Vec3(0.0036, -2.5452, -9.4741), Vec3(0.0081, -2.5516, -9.4723),
 Vec3(0.0155, -2.5611, -9.4698), Vec3(0.0266, -2.5737, -9.4663),
 Vec3(0.0421, -2.5895, -9.4620), Vec3(0.0623, -2.6082, -9.4567),
 Vec3(0.0876, -2.6293, -9.4507), Vec3(0.1180, -2.6519, -9.4440),
 Vec3(0.1533, -2.6753, -9.4369), Vec3(0.1928, -2.6984, -9.4296),
 Vec3(0.2359, -2.7201, -9.4224), Vec3(0.2814, -2.7393, -9.4156),
 Vec3(0.3284, -2.7548, -9.4095), Vec3(0.3753, -2.7656, -9.4046),
 Vec3(0.4209, -2.7709, -9.4011), Vec3(0.4636, -2.7699, -9.3994),
 Vec3(0.5021, -2.7621, -9.3997), Vec3(0.5349, -2.7470, -9.4023),
 Vec3(0.5606, -2.7244, -9.4074), Vec3(0.5777, -2.6942, -9.4151),
 Vec3(0.5848, -2.6563, -9.4254), Vec3(0.5805, -2.6109, -9.4383),
 Vec3(0.5633, -2.5583, -9.4538), Vec3(0.5318, -2.4988, -9.4715),
 Vec3(0.4849, -2.4331, -9.4911), Vec3(0.4215, -2.3621, -9.5120),
 Vec3(0.3414, -2.2872, -9.5335), Vec3(0.2446, -2.2101, -9.5547),
 Vec3(0.1320, -2.1330, -9.5744), Vec3(0.0055, -2.0586, -9.5916),
 Vec3(-0.1325, -1.9897, -9.6052), Vec3(-0.2784, -1.9297, -9.6143),
 Vec3(-0.4281, -1.8816, -9.6183), Vec3(-0.5771, -1.8482, -9.6170),
 Vec3(-0.7202, -1.8316, -9.6105), Vec3(-0.8524, -1.8330, -9.5995),
 Vec3(-0.9688, -1.8526, -9.5846), Vec3(-1.0647, -1.8896, -9.5672),
 Vec3(-1.1362, -1.9425, -9.5484), Vec3(-1.1803, -2.0096, -9.5291),
 Vec3(-1.1950, -2.0886, -9.5103), Vec3(-1.1793, -2.1774, -9.4923),
 Vec3(-1.1333, -2.2739, -9.4753), Vec3(-1.0581, -2.3756, -9.4590),
 Vec3(-0.9558, -2.4800, -9.4431), Vec3(-0.8291, -2.5843, -9.4271),
 Vec3(-0.6816, -2.6854, -9.4107), Vec3(-0.5172, -2.7799, -9.3936),
 Vec3(-0.3405, -2.8648, -9.3762), Vec3(-0.1559, -2.9367, -9.3588),
 Vec3(0.0319, -2.9931, -9.3422), Vec3(0.2186, -3.0320, -9.3271),
 Vec3(0.4003, -3.0526, -9.3144), Vec3(0.5739, -3.0549, -9.3045),
 Vec3(0.7368, -3.0402, -9.2979), Vec3(0.8871, -3.0106, -9.2944),
 Vec3(1.0230, -2.9683, -9.2940), Vec3(1.1431, -2.9152, -9.2968),
 Vec3(1.2455, -2.8524, -9.3032), Vec3(1.3279, -2.7796, -9.3138),
 Vec3(1.3873, -2.6957, -9.3298), Vec3(1.4202, -2.5991, -9.3522),
 Vec3(1.4224, -2.4880, -9.3820), Vec3(1.3898, -2.3612, -9.4196),
 Vec3(1.3187, -2.2185, -9.4644), Vec3(1.2066, -2.0610, -9.5149),
 Vec3(1.0524, -1.8913, -9.5683), Vec3(0.8575, -1.7140, -9.6210),
 Vec3(0.6258, -1.5351, -9.6689), Vec3(0.3634, -1.3625, -9.7081),
 Vec3(0.0786, -1.2053, -9.7354), Vec3(-0.2190, -1.0729, -9.7487),
 Vec3(-0.5192, -0.9734, -9.7478), Vec3(-0.8115, -0.9116, -9.7338),
 Vec3(-1.0858, -0.8878, -9.7092), Vec3(-1.3328, -0.8977, -9.6775),
 Vec3(-1.5445, -0.9352, -9.6424), Vec3(-1.7152, -0.9939, -9.6076),
 Vec3(-1.8417, -1.0686, -9.5761), Vec3(-1.9226, -1.1555, -9.5501),
 Vec3(-1.9578, -1.2518, -9.5308), Vec3(-1.9477, -1.3559, -9.5186),
 Vec3(-1.8937, -1.4665, -9.5131), Vec3(-1.7976, -1.5824, -9.5132),
 Vec3(-1.6618, -1.7021, -9.5172), Vec3(-1.4897, -1.8234, -9.5232),
 Vec3(-1.2852, -1.9435, -9.5293), Vec3(-1.0527, -2.0586, -9.5336),
 Vec3(-0.7974, -2.1650, -9.5348), Vec3(-0.5247, -2.2587, -9.5320),
 Vec3(-0.2405, -2.3361, -9.5247), Vec3(0.0494, -2.3946, -9.5131),
 Vec3(0.3390, -2.4325, -9.4976), Vec3(0.6226, -2.4494, -9.4789),
 Vec3(0.8946, -2.4455, -9.4581), Vec3(1.1497, -2.4222, -9.4365),
 Vec3(1.3828, -2.3809, -9.4157), Vec3(1.5890, -2.3227, -9.3977),
 Vec3(1.7635, -2.2480, -9.3847), Vec3(1.9019, -2.1566, -9.3791),
 Vec3(2.0000, -2.0478, -9.3831), Vec3(2.0543, -1.9207, -9.3982),
 Vec3(2.0619, -1.7747, -9.4252), Vec3(2.0209, -1.6106, -9.4635),

```

    Vec3(1.9307, -1.4302, -9.5112),  Vec3(1.7925, -1.2375, -9.5651),
    Vec3(1.6091, -1.0380, -9.6213),  Vec3(1.3851, -0.8392, -9.6754),
    Vec3(1.1271, -0.6497, -9.7234),  Vec3(0.8429, -0.4785, -9.7620),
    Vec3(0.5412, -0.3339, -9.7894),  Vec3(0.2310, -0.2222, -9.8048),
    Vec3(-0.0790, -0.1461, -9.8086), Vec3(-0.3805, -0.1046, -9.8021),
    Vec3(-0.6660, -0.0940, -9.7869), Vec3(-0.9288, -0.1088, -9.7653),
    Vec3(-1.1635, -0.1437, -9.7397), Vec3(-1.3659, -0.1941, -9.7125),
    Vec3(-1.5328, -0.2560, -9.6861),  Vec3(-1.6618, -0.3267, -9.6627),
    Vec3(-1.7513, -0.4039, -9.6440),  Vec3(-1.8004, -0.4861, -9.6311),
    Vec3(-1.8088, -0.5719, -9.6248),  Vec3(-1.7771, -0.6603, -9.6251),
    Vec3(-1.7065, -0.7500, -9.6313),  Vec3(-1.5988, -0.8396, -9.6424),
    Vec3(-1.4566, -0.9274, -9.6568),  Vec3(-1.2833, -1.0116, -9.6730),
    Vec3(-1.0826, -1.0901, -9.6890),  Vec3(-0.8591, -1.1608, -9.7031),
    Vec3(-0.6178, -1.2217, -9.7140),  Vec3(-0.3642, -1.2713, -9.7205),
    Vec3(-0.1042, -1.3082, -9.7218),  Vec3(0.1563, -1.3313, -9.7180),
    Vec3(0.4112, -1.3400, -9.7093),  Vec3(0.6546, -1.3340, -9.6968),
    Vec3(0.8810, -1.3132, -9.6817),  Vec3(1.0854, -1.2778, -9.6657),
    Vec3(1.2636, -1.2281, -9.6504),  Vec3(1.4122, -1.1650, -9.6377),
    Vec3(1.5285, -1.0894, -9.6288),  Vec3(1.6109, -1.0028, -9.6247),
    Vec3(1.6587, -0.9071, -9.6261),  Vec3(1.6719, -0.8046, -9.6329),
    Vec3(1.6513, -0.6979, -9.6448),  Vec3(1.5984, -0.5901, -9.6609),
    Vec3(1.5154, -0.4841, -9.6801),  Vec3(1.4049, -0.3830, -9.7013),
    Vec3(1.2703, -0.2897, -9.7231),  Vec3(1.1150, -0.2066, -9.7442),
    Vec3(0.9433, -0.1361, -9.7636),  Vec3(0.7593, -0.0795, -9.7803),
    Vec3(0.5675, -0.0381, -9.7935),  Vec3(0.3725, -0.0121, -9.8029),
    Vec3(0.1787, -0.0016, -9.8084),  Vec3(-0.0095, -0.0059, -9.8100),
    Vec3(-0.1881, -0.0239, -9.8082), Vec3(-0.3535, -0.0539, -9.8035),
    Vec3(-0.5027, -0.0943, -9.7967), Vec3(-0.6330, -0.1429, -9.7885),
    Vec3(-0.7425, -0.1974, -9.7799), Vec3(-0.8300, -0.2556, -9.7715),
    Vec3(-0.8948, -0.3153, -9.7640), Vec3(-0.9368, -0.3743, -9.7580),
    Vec3(-0.9566, -0.4308, -9.7537), Vec3(-0.9550, -0.4831, -9.7514),
    Vec3(-0.9336, -0.5300, -9.7511), Vec3(-0.8940, -0.5705, -9.7525),
    Vec3(-0.8380, -0.6042, -9.7554), Vec3(-0.7678, -0.6310, -9.7595),
    Vec3(-0.6853, -0.6509, -9.7644), Vec3(-0.5925, -0.6647, -9.7695),
    Vec3(-0.4913, -0.6730, -9.7746), Vec3(-0.3836, -0.6767, -9.7791),
    Vec3(-0.2712, -0.6769, -9.7829), Vec3(-0.1557, -0.6743, -9.7856),
    Vec3(-0.0389, -0.6697, -9.7870), Vec3(0.0776, -0.6633, -9.7872),
    Vec3(0.1922, -0.6555, -9.7862), Vec3(0.3034, -0.6462, -9.7840),
    Vec3(0.4097, -0.6353, -9.7808), Vec3(0.5099, -0.6225, -9.7769),
    Vec3(0.6027, -0.6078, -9.7726), Vec3(0.6870, -0.5910, -9.7681),
    Vec3(0.7617, -0.5725, -9.7636), Vec3(0.8260, -0.5527, -9.7595),
    Vec3(0.8790, -0.5323, -9.7560), Vec3(0.9202, -0.5123, -9.7533),
    Vec3(0.9491, -0.4937, -9.7515), Vec3(0.9656, -0.4775, -9.7507),
    Vec3(0.9698, -0.4646, -9.7509), Vec3(0.9623, -0.4556, -9.7521),
    Vec3(0.9436, -0.4508, -9.7541), Vec3(0.9148, -0.4502, -9.7569),
    Vec3(0.8768, -0.4531, -9.7602), Vec3(0.8310, -0.4590, -9.7640),
    Vec3(0.7787, -0.4667, -9.7679)};;

```

```

const Vec3 AngularVelocities2[] = {Vec3(0.0000, 0.0000, 0.0000),     Vec3(-0.0002, 0.0006, -0.0003),
                                  Vec3(-0.0015, 0.0042, -0.0029),  Vec3(-0.0046, 0.0122, -0.0098),
                                  Vec3(-0.0101, 0.0255, -0.0230),  Vec3(-0.0181, 0.0444, -0.0436),
                                  Vec3(-0.0288, 0.0687, -0.0721),  Vec3(-0.0422, 0.0979, -0.1083),
                                  Vec3(-0.0580, 0.1311, -0.1516),  Vec3(-0.0758, 0.1671, -0.2006),
                                  Vec3(-0.0949, 0.2044, -0.2533),  Vec3(-0.1149, 0.2416, -0.3074),
                                  Vec3(-0.1348, 0.2768, -0.3605),  Vec3(-0.1540, 0.3082, -0.4096),
                                  Vec3(-0.1715, 0.3339, -0.4519),  Vec3(-0.1864, 0.3521, -0.4845),
                                  Vec3(-0.1977, 0.3607, -0.5047),  Vec3(-0.2047, 0.3580, -0.5101),
                                  Vec3(-0.2065, 0.3423, -0.4984),  Vec3(-0.2023, 0.3122, -0.4681),
                                  Vec3(-0.1914, 0.2667, -0.4182),  Vec3(-0.1735, 0.2055, -0.3484),
                                  Vec3(-0.1482, 0.1287, -0.2592),  Vec3(-0.1155, 0.0375, -0.1521),
                                  Vec3(-0.0756, -0.0659, -0.0291), Vec3(-0.0289, -0.1786, 0.1069),
                                  Vec3(0.0237, -0.2965, 0.2524),   Vec3(0.0814, -0.4152, 0.4038),
                                  Vec3(0.1426, -0.5294, 0.5573),   Vec3(0.2055, -0.6342, 0.7093),
                                  Vec3(0.2677, -0.7246, 0.8562),   Vec3(0.3263, -0.7963, 0.9945),
                                  Vec3(0.3782, -0.8460, 1.1206),   Vec3(0.4198, -0.8713, 1.2307),
                                  Vec3(0.4481, -0.8710, 1.3202),   Vec3(0.4603, -0.8453, 1.3840),

```

$\text{Vec3}(0.4552, -0.7954, 1.4164), \text{Vec3}(0.4329, -0.7235, 1.4115),$
 $\text{Vec3}(0.3952, -0.6325, 1.3640), \text{Vec3}(0.3454, -0.5259, 1.2706),$
 $\text{Vec3}(0.2876, -0.4070, 1.1310), \text{Vec3}(0.2257, -0.2798, 0.9489),$
 $\text{Vec3}(0.1624, -0.1479, 0.7312), \text{Vec3}(0.0993, -0.0150, 0.4875),$
 $\text{Vec3}(0.0365, 0.1158, 0.2284), \text{Vec3}(-0.0267, 0.2420, -0.0358),$
 $\text{Vec3}(-0.0913, 0.3618, -0.2962), \text{Vec3}(-0.1581, 0.4744, -0.5454),$
 $\text{Vec3}(-0.2275, 0.5792, -0.7776), \text{Vec3}(-0.2994, 0.6760, -0.9884),$
 $\text{Vec3}(-0.3728, 0.7646, -1.1743), \text{Vec3}(-0.4465, 0.8444, -1.3329),$
 $\text{Vec3}(-0.5187, 0.9147, -1.4617), \text{Vec3}(-0.5872, 0.9739, -1.5583),$
 $\text{Vec3}(-0.6495, 1.0199, -1.6201), \text{Vec3}(-0.7026, 1.0499, -1.6441),$
 $\text{Vec3}(-0.7432, 1.0601, -1.6271), \text{Vec3}(-0.7678, 1.0462, -1.5663),$
 $\text{Vec3}(-0.7731, 1.0028, -1.4598), \text{Vec3}(-0.7567, 0.9247, -1.3075),$
 $\text{Vec3}(-0.7178, 0.8074, -1.1116), \text{Vec3}(-0.6574, 0.6486, -0.8772),$
 $\text{Vec3}(-0.5777, 0.4494, -0.6119), \text{Vec3}(-0.4818, 0.2148, -0.3250),$
 $\text{Vec3}(-0.3722, -0.0457, -0.0264), \text{Vec3}(-0.2509, -0.3199, 0.2741),$
 $\text{Vec3}(-0.1191, -0.5935, 0.5686), \text{Vec3}(0.0215, -0.8524, 0.8515),$
 $\text{Vec3}(0.1683, -1.0833, 1.1198), \text{Vec3}(0.3170, -1.2752, 1.3731),$
 $\text{Vec3}(0.4608, -1.4196, 1.6121), \text{Vec3}(0.5909, -1.5111, 1.8380),$
 $\text{Vec3}(0.6965, -1.5471, 2.0497), \text{Vec3}(0.7664, -1.5287, 2.2428),$
 $\text{Vec3}(0.7906, -1.4598, 2.4071), \text{Vec3}(0.7638, -1.3474, 2.5251),$
 $\text{Vec3}(0.6888, -1.2000, 2.5734), \text{Vec3}(0.5787, -1.0258, 2.5270),$
 $\text{Vec3}(0.4541, -0.8317, 2.3703), \text{Vec3}(0.3359, -0.6251, 2.1082),$
 $\text{Vec3}(0.2367, -0.4162, 1.7676), \text{Vec3}(0.1596, -0.2166, 1.3847),$
 $\text{Vec3}(0.1013, -0.0352, 0.9899), \text{Vec3}(0.0550, 0.1241, 0.6018),$
 $\text{Vec3}(0.0137, 0.2612, 0.2298), \text{Vec3}(-0.0292, 0.3777, -0.1208),$
 $\text{Vec3}(-0.0789, 0.4761, -0.4454), \text{Vec3}(-0.1386, 0.5586, -0.7396),$
 $\text{Vec3}(-0.2092, 0.6272, -0.9991), \text{Vec3}(-0.2899, 0.6839, -1.2207),$
 $\text{Vec3}(-0.3780, 0.7299, -1.4024), \text{Vec3}(-0.4701, 0.7659, -1.5435),$
 $\text{Vec3}(-0.5623, 0.7917, -1.6438), \text{Vec3}(-0.6508, 0.8061, -1.7037),$
 $\text{Vec3}(-0.7316, 0.8075, -1.7233), \text{Vec3}(-0.8008, 0.7937, -1.7028),$
 $\text{Vec3}(-0.8548, 0.7621, -1.6427), \text{Vec3}(-0.8895, 0.7104, -1.5437),$
 $\text{Vec3}(-0.9016, 0.6363, -1.4080), \text{Vec3}(-0.8880, 0.5380, -1.2391),$
 $\text{Vec3}(-0.8470, 0.4151, -1.0418), \text{Vec3}(-0.7784, 0.2684, -0.8222),$
 $\text{Vec3}(-0.6832, 0.1008, -0.5872), \text{Vec3}(-0.5639, -0.0824, -0.3431),$
 $\text{Vec3}(-0.4242, -0.2741, -0.0954), \text{Vec3}(-0.2686, -0.4655, 0.1519),$
 $\text{Vec3}(-0.1022, -0.6469, 0.3966), \text{Vec3}(0.0686, -0.8089, 0.6375),$
 $\text{Vec3}(0.2365, -0.9429, 0.8745), \text{Vec3}(0.3929, -1.0420, 1.1073),$
 $\text{Vec3}(0.5284, -1.1015, 1.3342), \text{Vec3}(0.6336, -1.1194, 1.5513),$
 $\text{Vec3}(0.6998, -1.0963, 1.7515), \text{Vec3}(0.7213, -1.0360, 1.9230),$
 $\text{Vec3}(0.6972, -0.9438, 2.0503), \text{Vec3}(0.6334, -0.8267, 2.1161),$
 $\text{Vec3}(0.5421, -0.6915, 2.1057), \text{Vec3}(0.4392, -0.5447, 2.0135),$
 $\text{Vec3}(0.3392, -0.3935, 1.8470), \text{Vec3}(0.2518, -0.2458, 1.6248),$
 $\text{Vec3}(0.1809, -0.1091, 1.3692), \text{Vec3}(0.1258, 0.0114, 1.0991),$
 $\text{Vec3}(0.0834, 0.1129, 0.8276), \text{Vec3}(0.0494, 0.1951, 0.5623),$
 $\text{Vec3}(0.0192, 0.2587, 0.3082), \text{Vec3}(-0.0112, 0.3051, 0.0692),$
 $\text{Vec3}(-0.0452, 0.3357, -0.1513), \text{Vec3}(-0.0848, 0.3522, -0.3499),$
 $\text{Vec3}(-0.1307, 0.3560, -0.5232), \text{Vec3}(-0.1824, 0.3485, -0.6687),$
 $\text{Vec3}(-0.2381, 0.3310, -0.7845), \text{Vec3}(-0.2954, 0.3044, -0.8696),$
 $\text{Vec3}(-0.3513, 0.2696, -0.9241), \text{Vec3}(-0.4025, 0.2272, -0.9489),$
 $\text{Vec3}(-0.4460, 0.1780, -0.9458), \text{Vec3}(-0.4790, 0.1227, -0.9174),$
 $\text{Vec3}(-0.4991, 0.0624, -0.8668), \text{Vec3}(-0.5045, -0.0013, -0.7975),$
 $\text{Vec3}(-0.4941, -0.0666, -0.7135), \text{Vec3}(-0.4677, -0.1314, -0.6187),$
 $\text{Vec3}(-0.4259, -0.1929, -0.5169), \text{Vec3}(-0.3699, -0.2488, -0.4116),$
 $\text{Vec3}(-0.3020, -0.2963, -0.3057), \text{Vec3}(-0.2250, -0.3334, -0.2017),$
 $\text{Vec3}(-0.1424, -0.3581, -0.1014), \text{Vec3}(-0.0577, -0.3694, -0.0060),$
 $\text{Vec3}(0.0255, -0.3670, 0.0833), \text{Vec3}(0.1038, -0.3511, 0.1658),$
 $\text{Vec3}(0.1744, -0.3229, 0.2406), \text{Vec3}(0.2350, -0.2840, 0.3069),$
 $\text{Vec3}(0.2840, -0.2365, 0.3640), \text{Vec3}(0.3208, -0.1827, 0.4110),$
 $\text{Vec3}(0.3453, -0.1252, 0.4473), \text{Vec3}(0.3580, -0.0664, 0.4723),$
 $\text{Vec3}(0.3598, -0.0088, 0.4862), \text{Vec3}(0.3521, 0.0453, 0.4894),$
 $\text{Vec3}(0.3361, 0.0940, 0.4828), \text{Vec3}(0.3133, 0.1354, 0.4679),$
 $\text{Vec3}(0.2851, 0.1681, 0.4465), \text{Vec3}(0.2525, 0.1908, 0.4210),$
 $\text{Vec3}(0.2166, 0.2027, 0.3934), \text{Vec3}(0.1784, 0.2033, 0.3661),$
 $\text{Vec3}(0.1387, 0.1922, 0.3413), \text{Vec3}(0.0985, 0.1697, 0.3207),$
 $\text{Vec3}(0.0587, 0.1362, 0.3059), \text{Vec3}(0.0202, 0.0925, 0.2979),$
 $\text{Vec3}(-0.0160, 0.0397, 0.2973), \text{Vec3}(-0.0487, -0.0206, 0.3038),$

```

    Vec3(-0.0770, -0.0867, 0.3170),  Vec3(-0.1000, -0.1563, 0.3356),
    Vec3(-0.1172, -0.2268, 0.3578),  Vec3(-0.1280, -0.2953, 0.3813),
    Vec3(-0.1326, -0.3588, 0.4034),  Vec3(-0.1314, -0.4142, 0.4213),
    Vec3(-0.1251, -0.4586, 0.4320),  Vec3(-0.1149, -0.4896, 0.4328),
    Vec3(-0.1021, -0.5051, 0.4213),  Vec3(-0.0880, -0.5040, 0.3960),
    Vec3(-0.0738, -0.4859, 0.3556),  Vec3(-0.0604, -0.4511, 0.3000),
    Vec3(-0.0485, -0.4006, 0.2294),  Vec3(-0.0383, -0.3362, 0.1450),
    Vec3(-0.0297, -0.2597, 0.0482),  Vec3(-0.0224, -0.1735, -0.0589),
    Vec3(-0.0160, -0.0804, -0.1739), Vec3(-0.0100, 0.0169, -0.2938),
    Vec3(-0.0041, 0.1154, -0.4155),  Vec3(0.0021, 0.2120, -0.5354),
    Vec3(0.0085, 0.3039, -0.6501),  Vec3(0.0151, 0.3883, -0.7561),
    Vec3(0.0215, 0.4632, -0.8503),  Vec3(0.0273, 0.5267, -0.9301),
    Vec3(0.0323, 0.5775, -0.9933),  Vec3(0.0361, 0.6147, -1.0379),
    Vec3(0.0387, 0.6377, -1.0625),  Vec3(0.0399, 0.6460, -1.0657),
    Vec3(0.0399, 0.6391, -1.0462),  Vec3(0.0390, 0.6165, -1.0028),
    Vec3(0.0374, 0.5779, -0.9345),  Vec3(0.0357, 0.5230, -0.8407),
    Vec3(0.0342, 0.4520, -0.7211)};
}

```

```

const Vec3 Acceleration2[] = {Vec3(0.0000, 0.0000, -0.0000),  Vec3(-0.0040, 0.0002, -0.0021),
    Vec3(-0.0151, 0.0024, -0.0079),  Vec3(-0.0316, 0.0073, -0.0169),
    Vec3(-0.0517, 0.0149, -0.0280),  Vec3(-0.0736, 0.0246, -0.0404),
    Vec3(-0.0954, 0.0358, -0.0531),  Vec3(-0.1154, 0.0476, -0.0651),
    Vec3(-0.1320, 0.0590, -0.0757),  Vec3(-0.1437, 0.0689, -0.0839),
    Vec3(-0.1494, 0.0765, -0.0892),  Vec3(-0.1482, 0.0808, -0.0908),
    Vec3(-0.1393, 0.0813, -0.0883),  Vec3(-0.1224, 0.0779, -0.0812),
    Vec3(-0.0975, 0.0706, -0.0692),  Vec3(-0.0651, 0.0599, -0.0522),
    Vec3(-0.0258, 0.0465, -0.0303),  Vec3(0.0192, 0.0314, -0.0036),
    Vec3(0.0686, 0.0154, 0.0271),  Vec3(0.1205, -0.0006, 0.0611),
    Vec3(0.1734, -0.0162, 0.0970),  Vec3(0.2255, -0.0311, 0.1335),
    Vec3(0.2754, -0.0457, 0.1687),  Vec3(0.3216, -0.0606, 0.2007),
    Vec3(0.3630, -0.0769, 0.2277),  Vec3(0.3984, -0.0958, 0.2479),
    Vec3(0.4264, -0.1185, 0.2602),  Vec3(0.4451, -0.1460, 0.2635),
    Vec3(0.4522, -0.1788, 0.2575),  Vec3(0.4448, -0.2167, 0.2423),
    Vec3(0.4203, -0.2583, 0.2183),  Vec3(0.3760, -0.3014, 0.1863),
    Vec3(0.3103, -0.3420, 0.1470),  Vec3(0.2230, -0.3753, 0.1015),
    Vec3(0.1159, -0.3954, 0.0510),  Vec3(-0.0068, -0.3961, -0.0033),
    Vec3(-0.1381, -0.3719, -0.0591), Vec3(-0.2686, -0.3185, -0.1139),
    Vec3(-0.3870, -0.2352, -0.1645), Vec3(-0.4828, -0.1248, -0.2081),
    Vec3(-0.5486, 0.0053, -0.2425),  Vec3(-0.5820, 0.1447, -0.2676),
    Vec3(-0.5869, 0.2817, -0.2850),  Vec3(-0.5719, 0.4063, -0.2972),
    Vec3(-0.5465, 0.5109, -0.3070),  Vec3(-0.5188, 0.5910, -0.3161),
    Vec3(-0.4937, 0.6443, -0.3248),  Vec3(-0.4722, 0.6704, -0.3321),
    Vec3(-0.4525, 0.6694, -0.3363),  Vec3(-0.4312, 0.6425, -0.3353),
    Vec3(-0.4039, 0.5915, -0.3269),  Vec3(-0.3663, 0.5190, -0.3093),
    Vec3(-0.3148, 0.4289, -0.2807),  Vec3(-0.2466, 0.3261, -0.2399),
    Vec3(-0.1609, 0.2170, -0.1860),  Vec3(-0.0588, 0.1089, -0.1190),
    Vec3(0.0562, 0.0096, -0.0397),  Vec3(0.1780, -0.0735, 0.0497),
    Vec3(0.2990, -0.1352, 0.1458),  Vec3(0.4117, -0.1736, 0.2436),
    Vec3(0.5108, -0.1901, 0.3375),  Vec3(0.5946, -0.1900, 0.4218),
    Vec3(0.6655, -0.1802, 0.4908),  Vec3(0.7280, -0.1681, 0.5401),
    Vec3(0.7863, -0.1611, 0.5668),  Vec3(0.8416, -0.1655, 0.5702),
    Vec3(0.8905, -0.1875, 0.5519),  Vec3(0.9260, -0.2324, 0.5158),
    Vec3(0.9383, -0.3036, 0.4666),  Vec3(0.9166, -0.4017, 0.4091),
    Vec3(0.8505, -0.5234, 0.3470),  Vec3(0.7312, -0.6613, 0.2816),
    Vec3(0.5517, -0.8032, 0.2122),  Vec3(0.3092, -0.9318, 0.1363),
    Vec3(0.0076, -1.0243, 0.0511),  Vec3(-0.3365, -1.0523, -0.0440),
    Vec3(-0.6861, -0.9865, -0.1433), Vec3(-0.9811, -0.8083, -0.2331),
    Vec3(-1.1530, -0.5276, -0.2965), Vec3(-1.1643, -0.1908, -0.3262),
    Vec3(-1.0394, 0.1408, -0.3317), Vec3(-0.8471, 0.4235, -0.3310),
    Vec3(-0.6532, 0.6443, -0.3369), Vec3(-0.4942, 0.8086, -0.3533),
    Vec3(-0.3818, 0.9256, -0.3783), Vec3(-0.3134, 1.0026, -0.4079),
    Vec3(-0.2806, 1.0436, -0.4375), Vec3(-0.2726, 1.0514, -0.4622),
    Vec3(-0.2782, 1.0279, -0.4772), Vec3(-0.2876, 0.9756, -0.4787),
    Vec3(-0.2929, 0.8975, -0.4641), Vec3(-0.2884, 0.7970, -0.4322),
    Vec3(-0.2703, 0.6784, -0.3830), Vec3(-0.2361, 0.5465, -0.3175),
    Vec3(-0.1845, 0.4070, -0.2375), Vec3(-0.1154, 0.2662, -0.1458),
}

```

```

    Vec3(-0.0301, 0.1306, -0.0459), Vec3(0.0686, 0.0063, 0.0575),
    Vec3(0.1770, -0.1019, 0.1593), Vec3(0.2907, -0.1909, 0.2541),
    Vec3(0.4050, -0.2603, 0.3372), Vec3(0.5155, -0.3119, 0.4050),
    Vec3(0.6180, -0.3497, 0.4557), Vec3(0.7084, -0.3789, 0.4890),
    Vec3(0.7826, -0.4057, 0.5068), Vec3(0.8362, -0.4358, 0.5118),
    Vec3(0.8647, -0.4744, 0.5075), Vec3(0.8634, -0.5249, 0.4970),
    Vec3(0.8277, -0.5888, 0.4829), Vec3(0.7532, -0.6652, 0.4659),
    Vec3(0.6357, -0.7496, 0.4454), Vec3(0.4724, -0.8343, 0.4189),
    Vec3(0.2635, -0.9069, 0.3827), Vec3(0.0155, -0.9505, 0.3327),
    Vec3(-0.2548, -0.9452, 0.2661), Vec3(-0.5169, -0.8722, 0.1838),
    Vec3(-0.7292, -0.7231, 0.0912), Vec3(-0.8531, -0.5082, -0.0034),
    Vec3(-0.8724, -0.2568, -0.0932), Vec3(-0.8030, -0.0046, -0.1753),
    Vec3(-0.6802, 0.2220, -0.2500), Vec3(-0.5386, 0.4110, -0.3182),
    Vec3(-0.4015, 0.5610, -0.3799), Vec3(-0.2808, 0.6748, -0.4341),
    Vec3(-0.1816, 0.7563, -0.4799), Vec3(-0.1050, 0.8086, -0.5161),
    Vec3(-0.0502, 0.8347, -0.5415), Vec3(-0.0146, 0.8372, -0.5548),
    Vec3(0.0053, 0.8186, -0.5550), Vec3(0.0135, 0.7813, -0.5411),
    Vec3(0.0139, 0.7279, -0.5129), Vec3(0.0100, 0.6609, -0.4710),
    Vec3(0.0050, 0.5827, -0.4162), Vec3(0.0012, 0.4956, -0.3505),
    Vec3(0.0006, 0.4020, -0.2761), Vec3(0.0043, 0.3040, -0.1954),
    Vec3(0.0131, 0.2038, -0.1114), Vec3(0.0273, 0.1035, -0.0266),
    Vec3(0.0465, 0.0050, 0.0564), Vec3(0.0699, -0.0897, 0.1358),
    Vec3(0.0961, -0.1785, 0.2098), Vec3(0.1231, -0.2599, 0.2774),
    Vec3(0.1488, -0.3321, 0.3380), Vec3(0.1707, -0.3937, 0.3910),
    Vec3(0.1864, -0.4436, 0.4361), Vec3(0.1939, -0.4810, 0.4728),
    Vec3(0.1916, -0.5055, 0.5005), Vec3(0.1785, -0.5167, 0.5185),
    Vec3(0.1545, -0.5150, 0.5257), Vec3(0.1206, -0.5007, 0.5212),
    Vec3(0.0787, -0.4745, 0.5042), Vec3(0.0315, -0.4373, 0.4741),
    Vec3(-0.0175, -0.3904, 0.4311), Vec3(-0.0646, -0.3354, 0.3757),
    Vec3(-0.1061, -0.2740, 0.3093), Vec3(-0.1385, -0.2084, 0.2339),
    Vec3(-0.1593, -0.1409, 0.1520), Vec3(-0.1669, -0.0740, 0.0666),
    Vec3(-0.1608, -0.0099, -0.0191), Vec3(-0.1419, 0.0491, -0.1021),
    Vec3(-0.1118, 0.1015, -0.1797), Vec3(-0.0730, 0.1458, -0.2495),
    Vec3(-0.0288, 0.1813, -0.3098), Vec3(0.0175, 0.2074, -0.3595),
    Vec3(0.0625, 0.2243, -0.3981), Vec3(0.1027, 0.2323, -0.4257),
    Vec3(0.1355, 0.2321, -0.4426), Vec3(0.1583, 0.2244, -0.4495),
    Vec3(0.1696, 0.2104, -0.4471), Vec3(0.1686, 0.1910, -0.4359),
    Vec3(0.1553, 0.1677, -0.4164), Vec3(0.1306, 0.1416, -0.3889),
    Vec3(0.0964, 0.1144, -0.3535), Vec3(0.0557, 0.0876, -0.3101),
    Vec3(0.0118, 0.0626, -0.2591), Vec3(-0.0315, 0.0410, -0.2010),
    Vec3(-0.0708, 0.0239, -0.1369), Vec3(-0.1035, 0.0123, -0.0685),
    Vec3(-0.1280, 0.0066, 0.0020), Vec3(-0.1443, 0.0071, 0.0721),
    Vec3(-0.1536, 0.0133, 0.1393), Vec3(-0.1582, 0.0246, 0.2011),
    Vec3(-0.1607, 0.0400, 0.2556), Vec3(-0.1640, 0.0583, 0.3011),
    Vec3(-0.1701, 0.0778, 0.3363), Vec3(-0.1802, 0.0970, 0.3604),
    Vec3(-0.1943, 0.1142, 0.3728), Vec3(-0.2108, 0.1280, 0.3735),
    Vec3(-0.2273, 0.1370, 0.3627), Vec3(-0.2405, 0.1405, 0.3411),
    Vec3(-0.2468, 0.1380, 0.3101), Vec3(-0.2431, 0.1298, 0.2713),
    Vec3(-0.2271, 0.1162, 0.2267), Vec3(-0.1974, 0.0982, 0.1784),
    Vec3(-0.1536, 0.0769, 0.1291), Vec3(-0.0963, 0.0534, 0.0808),
    Vec3(-0.0270, 0.0290, 0.0359), Vec3(0.0519, 0.0049, -0.0037),
    Vec3(0.1376, -0.0183, -0.0368), Vec3(0.2265, -0.0402, -0.0625),
    Vec3(0.3146, -0.0611, -0.0808)};
}

const Vec3 Gravity2[] = {Vec3(-6.9367, 3.4684, -6.0074), Vec3(-6.9368, 3.4684, -6.0073),
    Vec3(-6.9371, 3.4686, -6.0067), Vec3(-6.9385, 3.4698, -6.0045),
    Vec3(-6.9412, 3.4731, -5.9994), Vec3(-6.9459, 3.4803, -5.9898),
    Vec3(-6.9527, 3.4932, -5.9744), Vec3(-6.9618, 3.5138, -5.9517),
    Vec3(-6.9728, 3.5441, -5.9207), Vec3(-6.9855, 3.5857, -5.8806),
    Vec3(-6.9989, 3.6400, -5.8311), Vec3(-7.0121, 3.7078, -5.7722),
    Vec3(-7.0239, 3.7892, -5.7046), Vec3(-7.0330, 3.8838, -5.6294),
    Vec3(-7.0379, 3.9900, -5.5483), Vec3(-7.0376, 4.1058, -5.4636),
    Vec3(-7.0311, 4.2280, -5.3780), Vec3(-7.0181, 4.3529, -5.2947),
    Vec3(-6.9988, 4.4760, -5.2171), Vec3(-6.9740, 4.5923, -5.1488),
    Vec3(-6.9451, 4.6966, -5.0935), Vec3(-6.9141, 4.7835, -5.0546),
    Vec3(-6.8830, 4.8482, -5.0354), Vec3(-6.8538, 4.8859, -5.0387),
}

```

$\text{Vec3}(-6.8280, 4.8931, -5.0668), \text{Vec3}(-6.8063, 4.8667, -5.1211),$
 $\text{Vec3}(-6.7883, 4.8051, -5.2025), \text{Vec3}(-6.7726, 4.7078, -5.3108),$
 $\text{Vec3}(-6.7570, 4.5753, -5.4448), \text{Vec3}(-6.7383, 4.4096, -5.6024),$
 $\text{Vec3}(-6.7134, 4.2136, -5.7803), \text{Vec3}(-6.6794, 3.9909, -5.9745),$
 $\text{Vec3}(-6.6340, 3.7458, -6.1802), \text{Vec3}(-6.5761, 3.4828, -6.3923),$
 $\text{Vec3}(-6.5059, 3.2064, -6.6052), \text{Vec3}(-6.4248, 2.9214, -6.8135),$
 $\text{Vec3}(-6.3353, 2.6325, -7.0121), \text{Vec3}(-6.2413, 2.3451, -7.1960),$
 $\text{Vec3}(-6.1475, 2.0653, -7.3607), \text{Vec3}(-6.0595, 1.8000, -7.5019),$
 $\text{Vec3}(-5.9840, 1.5572, -7.6160), \text{Vec3}(-5.9280, 1.3450, -7.6997),$
 $\text{Vec3}(-5.8983, 1.1709, -7.7508), \text{Vec3}(-5.9005, 1.0415, -7.7676),$
 $\text{Vec3}(-5.9381, 0.9618, -7.7492), \text{Vec3}(-6.0120, 0.9350, -7.6953),$
 $\text{Vec3}(-6.1198, 0.9632, -7.6064), \text{Vec3}(-6.2565, 1.0469, -7.4830),$
 $\text{Vec3}(-6.4148, 1.1857, -7.3267), \text{Vec3}(-6.5853, 1.3783, -7.1393),$
 $\text{Vec3}(-6.7576, 1.6218, -6.9239), \text{Vec3}(-6.9206, 1.9121, -6.6847),$
 $\text{Vec3}(-7.0638, 2.2435, -6.4269), \text{Vec3}(-7.1777, 2.6084, -6.1574),$
 $\text{Vec3}(-7.2550, 2.9969, -5.8839), \text{Vec3}(-7.2914, 3.3974, -5.6151),$
 $\text{Vec3}(-7.2864, 3.7965, -5.3601), \text{Vec3}(-7.2439, 4.1797, -5.1276),$
 $\text{Vec3}(-7.1714, 4.5326, -4.9256), \text{Vec3}(-7.0802, 4.8415, -4.7609),$
 $\text{Vec3}(-6.9830, 5.0945, -4.6389), \text{Vec3}(-6.8924, 5.2820, -4.5641),$
 $\text{Vec3}(-6.8185, 5.3973, -4.5402), \text{Vec3}(-6.7673, 5.4361, -4.5705),$
 $\text{Vec3}(-6.7392, 5.3970, -4.6574), \text{Vec3}(-6.7290, 5.2815, -4.8023),$
 $\text{Vec3}(-6.7261, 5.0941, -5.0045), \text{Vec3}(-6.7166, 4.8425, -5.2606),$
 $\text{Vec3}(-6.6851, 4.5375, -5.5639), \text{Vec3}(-6.6179, 4.1922, -5.9046),$
 $\text{Vec3}(-6.5054, 3.8207, -6.2704), \text{Vec3}(-6.3435, 3.4365, -6.6473),$
 $\text{Vec3}(-6.1343, 3.0506, -7.0214), \text{Vec3}(-5.8858, 2.6697, -7.3801),$
 $\text{Vec3}(-5.6090, 2.2962, -7.7138), \text{Vec3}(-5.3165, 1.9288, -8.0157),$
 $\text{Vec3}(-5.0196, 1.5657, -8.2818), \text{Vec3}(-4.7291, 1.2090, -8.5094),$
 $\text{Vec3}(-4.4566, 0.8669, -8.6962), \text{Vec3}(-4.2167, 0.5524, -8.8403),$
 $\text{Vec3}(-4.0257, 0.2787, -8.9416), \text{Vec3}(-3.8975, 0.0544, -9.0024),$
 $\text{Vec3}(-3.8397, -0.1168, -9.0266), \text{Vec3}(-3.8536, -0.2344, -9.0184),$
 $\text{Vec3}(-3.9356, -0.2994, -8.9810), \text{Vec3}(-4.0785, -0.3130, -8.9165),$
 $\text{Vec3}(-4.2727, -0.2768, -8.8263), \text{Vec3}(-4.5071, -0.1933, -8.7112),$
 $\text{Vec3}(-4.7694, -0.0655, -8.5723), \text{Vec3}(-5.0472, 0.1023, -8.4114),$
 $\text{Vec3}(-5.3283, 0.3049, -8.2312), \text{Vec3}(-5.6018, 0.5363, -8.0354),$
 $\text{Vec3}(-5.8581, 0.7893, -7.8291), \text{Vec3}(-6.0896, 1.0562, -7.6182),$
 $\text{Vec3}(-6.2908, 1.3279, -7.4093), \text{Vec3}(-6.4584, 1.5949, -7.2098),$
 $\text{Vec3}(-6.5915, 1.8468, -7.0269), \text{Vec3}(-6.6913, 2.0737, -6.8675),$
 $\text{Vec3}(-6.7604, 2.2660, -6.7379), \text{Vec3}(-6.8020, 2.4158, -6.6432),$
 $\text{Vec3}(-6.8192, 2.5174, -6.5876), \text{Vec3}(-6.8134, 2.5676, -6.5743),$
 $\text{Vec3}(-6.7839, 2.5663, -6.6052), \text{Vec3}(-6.7280, 2.5161, -6.6813),$
 $\text{Vec3}(-6.6407, 2.4227, -6.8020), \text{Vec3}(-6.5164, 2.2940, -6.9649),$
 $\text{Vec3}(-6.3498, 2.1403, -7.1649), \text{Vec3}(-6.1371, 1.9729, -7.3946),$
 $\text{Vec3}(-5.8779, 1.8036, -7.6442), \text{Vec3}(-5.5761, 1.6428, -7.9022),$
 $\text{Vec3}(-5.2400, 1.4987, -8.1567), \text{Vec3}(-4.8824, 1.3754, -8.3968),$
 $\text{Vec3}(-4.5190, 1.2723, -8.6137), \text{Vec3}(-4.1663, 1.1843, -8.8020),$
 $\text{Vec3}(-3.8394, 1.1028, -8.9598), \text{Vec3}(-3.5500, 1.0189, -9.0882),$
 $\text{Vec3}(-3.3056, 0.9257, -9.1898), \text{Vec3}(-3.1104, 0.8212, -9.2675),$
 $\text{Vec3}(-2.9665, 0.7080, -9.3239), \text{Vec3}(-2.8741, 0.5918, -9.3608),$
 $\text{Vec3}(-2.8317, 0.4789, -9.3802), \text{Vec3}(-2.8359, 0.3756, -9.3836),$
 $\text{Vec3}(-2.8818, 0.2867, -9.3728), \text{Vec3}(-2.9634, 0.2163, -9.3492),$
 $\text{Vec3}(-3.0741, 0.1669, -9.3144), \text{Vec3}(-3.2072, 0.1396, -9.2699),$
 $\text{Vec3}(-3.3555, 0.1339, -9.2173), \text{Vec3}(-3.5123, 0.1478, -9.1585),$
 $\text{Vec3}(-3.6707, 0.1778, -9.0956), \text{Vec3}(-3.8248, 0.2193, -9.0310),$
 $\text{Vec3}(-3.9691, 0.2670, -8.9672), \text{Vec3}(-4.0993, 0.3153, -8.9069),$
 $\text{Vec3}(-4.2119, 0.3586, -8.8526), \text{Vec3}(-4.3041, 0.3920, -8.8066),$
 $\text{Vec3}(-4.3743, 0.4114, -8.7711), \text{Vec3}(-4.4210, 0.4138, -8.7476),$
 $\text{Vec3}(-4.4434, 0.3977, -8.7370), \text{Vec3}(-4.4409, 0.3631, -8.7397),$
 $\text{Vec3}(-4.4132, 0.3117, -8.7557), \text{Vec3}(-4.3604, 0.2466, -8.7842),$
 $\text{Vec3}(-4.2828, 0.1723, -8.8241), \text{Vec3}(-4.1817, 0.0942, -8.8736),$
 $\text{Vec3}(-4.0590, 0.0182, -8.9309), \text{Vec3}(-3.9176, -0.0495, -8.9936),$
 $\text{Vec3}(-3.7617, -0.1034, -9.0595), \text{Vec3}(-3.5961, -0.1390, -9.1261),$
 $\text{Vec3}(-3.4267, -0.1531, -9.1908), \text{Vec3}(-3.2597, -0.1439, -9.2515),$
 $\text{Vec3}(-3.1016, -0.1113, -9.3061), \text{Vec3}(-2.9585, -0.0570, -9.3531),$
 $\text{Vec3}(-2.8359, 0.0165, -9.3911), \text{Vec3}(-2.7383, 0.1051, -9.4195),$
 $\text{Vec3}(-2.6689, 0.2046, -9.4377), \text{Vec3}(-2.6296, 0.3101, -9.4459),$
 $\text{Vec3}(-2.6207, 0.4170, -9.4443), \text{Vec3}(-2.6409, 0.5212, -9.4335),$

```
Vec3(-2.6879, 0.6188, -9.4143), Vec3(-2.7579, 0.7069, -9.3878),
Vec3(-2.8465, 0.7832, -9.3552), Vec3(-2.9484, 0.8460, -9.3181),
Vec3(-3.0580, 0.8940, -9.2782), Vec3(-3.1697, 0.9264, -9.2375),
Vec3(-3.2779, 0.9426, -9.1980), Vec3(-3.3770, 0.9420, -9.1621),
Vec3(-3.4620, 0.9245, -9.1322), Vec3(-3.5282, 0.8897, -9.1102),
Vec3(-3.5715, 0.8378, -9.0983), Vec3(-3.5881, 0.7692, -9.0978),
Vec3(-3.5752, 0.6849, -9.1096), Vec3(-3.5302, 0.5864, -9.1340),
Vec3(-3.4517, 0.4759, -9.1703), Vec3(-3.3393, 0.3565, -9.2173),
Vec3(-3.1937, 0.2319, -9.2727), Vec3(-3.0170, 0.1062, -9.3340),
Vec3(-2.8131, -0.0164, -9.3980), Vec3(-2.5873, -0.1316, -9.4617),
Vec3(-2.3467, -0.2360, -9.5223), Vec3(-2.0993, -0.3270, -9.5772),
Vec3(-1.8541, -0.4031, -9.6247), Vec3(-1.6204, -0.4640, -9.6641),
Vec3(-1.4073, -0.5104, -9.6951), Vec3(-1.2230, -0.5439, -9.7183),
Vec3(-1.0749, -0.5660, -9.7345), Vec3(-0.9693, -0.5784, -9.7448),
Vec3(-0.9105, -0.5824, -9.7503), Vec3(-0.9017, -0.5782, -9.7513),
Vec3(-0.9441, -0.5653, -9.7481), Vec3(-1.0372, -0.5423, -9.7399),
Vec3(-1.1785, -0.5069, -9.7258), Vec3(-1.3640, -0.4565, -9.7040),
Vec3(-1.5877, -0.3883, -9.6729), Vec3(-1.8424, -0.3001, -9.6308),
Vec3(-2.1199, -0.1903, -9.5763), Vec3(-2.4113, -0.0587, -9.5089),
Vec3(-2.7075, 0.0937, -9.4285), Vec3(-2.9997, 0.2644, -9.3364),
Vec3(-3.2799, 0.4490, -9.2345), Vec3(-3.5407, 0.6423, -9.1262),
Vec3(-3.7763, 0.8372, -9.0152), Vec3(-3.9822, 1.0262, -8.9065),
Vec3(-4.1551, 1.2010, -8.8051});
```