Technische Universität München, Institute for Media Technology

Massachussets Institute of Technology, Media Lab

Prof. Dr.-Ing. Eckehard Steinbach
Prof. Joseph A. Paradiso, PhD

Diplomarbeit

Fusion of Six Dimensional Sensor Data using Physics Engines

Author:	Clemens Satzger
Matriculation Number:	2656892
Address:	Blumenstr.10
	82272 Moorenweis
Email:	clemens.satzger@gmail.com
Advisor:	Michael Lapinski (MIT)
Begin:	08/01/2009
End:	01/31/2010

Abstract

Fusion of 6 Dimensional Sensor Data using Physics Engines
Capture of high speed athletic motion has until now only been possible by using expensive high speed camera systems. This work presents a novel alternative in form of a wireless inertial measurement system. The sensor data from this system is combined using a physics engine andallows a higher resolution calculation of forces and torques in the joints than is conventionally available from optical tracking systems.

Acknowledgements

- Thanks to my parents to have raised me the way I am now.
- Thanks to my sister to help me in all my difficult decisions.
- Thanks to Sonia to have helped me in my first days here and to have an open ear for everything.
- Thanks to Markus to make up the link to Joe.
- Thanks to Prof. Steinbach to advise my thesis.
- Thanks to Joe to give me the opportunity to write this thesis.
- Thanks to Mike to give me so much support in all belongings.
- Thanks to Nanwei to be such an awesome friend!
- Thanks to Matt to be such a good friend all the time and to put me on one of his patent claims.
- Thanks to Jesse to help me with the optical camera system.
- Thanks to Mat for all his advices of implementation.
- Thanks to Bo for his programming tips.
- Thanks to Spinner to fill useless free time slots.
- Thanks to Boston Metro to keep me in shape.
- Thanks to Slayers van to keep Mike occupied.
- Thanks to WD and flux cleaner.

Contents

1. Introduction

Motion tracking also called video tracking by cameras is state of the art technology to import a movement in a virtual computer environment. However, it would be a lot more flexible to use low cost inertial sensors attached to the body. These sensors provide the opportunity to track movements at any place and during any time. Camera systems do not have this ability; they are expensive and need to be in a special environment. They have to be fixed and installed in a darkened room in order to track the movements of marker points at the object. The data of this movement is limited by the frame rate of the video cameras, however high speed cameras can have a frame rate of up to 400 Hz. An inertial sensor system allows much higher data rate, more directly measures biomechanical relevant features like forces and torques and is in the same time more flexible and cheaper. However, the data of the inertial measurement system need to be properly fused, filtered and processed to allow interpretations of the measures.

1.1 Synopsis

This thesis describes the handling and fusion of the data gathered from the sportSemble system described in Michael Lapinski's thesis [Lap08]. The further sections of the introduction describe the motivation and the prior work, which this thesis is based on. The chapter *The Concept of the Wireless IMU System* explains the hardware of the wireless inertial measurement system (IMU) and its calibration . The chapter *The Management of Data* describes the data storage. Next, the *Data Fusion* chapter goes through a physics engine as well as a mathematical Kalman filter structure that reduces the drift of the integrated data. After this the chapter *Results* presents the products of the data fusion. The final chapter *Conclusions* gives some suggestions about future improvements to the system.

1.2 Motivation

The motivation of this thesis can be structured in three areas.

- Medical relevance
- Improvement of training methods
- Evaluation of athletic performance

The first most obvious area is the medical relevance of this thesis. 50% of baseball pitchers have once in their career shoulder or elbow pain which keeps them from throwing [TH73]. During the last years this number even increased up to 75% by 1999 [CS01]. An unresolved problem in medicine is the analysis of stress in the joints of the bones. Camera based motion tracking systems obtain the positions of the body segments in the space, but it is really difficult to derive reliable forces from these position measurements as a double derivative over the time is necessary (typically 75% accurate [Ber07]). These forces as well as torques

in the joints of the bones are interesting in understanding injuries, as pitching injuries are a result of repetitive microtrauma [TH73, FG95, RL01, CK00, TJ72].

Not only the extreme forces, which cause injuries during athletic movements are interesting, but also using the inter-segmental forces and torques to refine technique and rehabilitation protocols is an interesting medical application.

The American Journal of Sports Medicine presents five parameters to improve pitching efficiency [JTDJ09]. As a result, good pitching mechanic considering these five parameters could reduce the stress in the arm without loosing pitching speed. The efficiency of a correct mechanical motion will reduce force and rotational torque. Another medical aspect is to detect the fatigue of an athlete, as lots of injuries are caused by loss of muscular control due to fatigue. If the athlete can be warned to be careful as his muscles are fatigued, he can react and step back to "safer" movements. Because "due to muscle fatigue and weakness, superior translation of the humeral head occurs, and the upper surface of the rotator cuff muscles and tendons is abraded against the under surface of the acromion" [FG96].

The second aspect is to improve the training of the Red Sox baseball team. The idea is that some players get equipped by the sensor system to be able to analyze their movement after the training. In this way the ideal movement can be found and also injured players can get back in shape faster after an injury. Furthermore, it can prevent pitchers form injury as their pitching data can be scanned regularly by the physicians. This allows to pitchers to be forewarned about their bad movements and followed by strengthening muscle groups before the shoulder is getting injured. It is a known fact, that most shoulder injuries in throwing involve soft tissues, including muscles, tendons, ligaments, capsule bursae, and fibrocartilaginous tissue [FG96].

The last motivation is the from the scouts, who want to find the best players out of a pool of players. Until now they only judged the movement of the players by watching them play, but with this sensor system their performance can be estimated much better. Therefore, this thesis is written using the input of the Red Sox scouts to understand what they are looking for to select players in a more scientific way.

1.2.1 Integration Issues

To understand the main advantages and drawbacks of inertial sensors for our application, we need to have a closer look at the differential equations involved. Camera systems directly output three dimensional (3D) coordinates of markers placed on athletes' bodies, whereas when tracking with inertial sensors, we have to integrate the linear acceleration twice to get a position. As a result of that integration, we get a second order error (Equation 1.1) that diverges quadratically for offset errors and linearly for Gaussian noise. Therefore, if the system goal is to produce a position, inertial systems are not typically as accurate as a camera-based motion tracking system.

$$v_k = v_{k-1} + \Delta t a_{k-1}$$
$$p_k = p_{k-1} + \Delta t v_{k-1} + \frac{\Delta t^2}{2} a_{k-1} \tag{1.1}$$
$$\varphi_k = \theta_{k-1} + \Delta t \omega_{k-1} \tag{1.2}$$

a_k	Acceleration vector in x,y,z
v_k	Velocity vector in x,y,z
p_k	Position vector in x,y,z
ω	Angular velocity vector in pitch, roll, yaw
φ	Angle vector in pitch, roll, yaw

In our case, although some level of position tracking is required, we are interested in accurately calculating the forces and the torques at the joints. The force at a body segment is essentially the weight of the segment times its acceleration. In this case the camera-based motion tracking system tends to be less accurate, because its measured displacements need to be doubly-differentiated to derive acceleration - first the position has to be differentiated to get the velocity of a segment, then this velocity has to be differentiated again to get the acceleration (differentiation is known to increase the level of noise in a signal [LM67]). An inertial sensor system, on the other hand, provides a direct measurement of acceleration, hence there is no need for integration or differentiation and the sampling rate and bandwidth of typical inertial systems can be, for example, over five times higher than the update rate of a standard motion-capture camera system.

1.3 Prior Work

1.3.1 Sports Medicine

As medical applications are one of the main targets for this inertial system is it interesting to have some background information about the causes of pitching injuries and why it is so important to be able to measure forces and torques in the joints. *Athletic Injuries and Rehabilitation* [FG96] describes the biomechanics of throwing in chapter 17, where it separates the throwing movement into six phases. The initial `windup phase` is when the player balances all his weight on one foot. This is followed by the `stride phase`, which is defined until the second foot of the player contacts the ground again. The subsequent `arm cocking phase` is from the second foot contact till the maximum external shoulder rotation. The `arm acceleration` phase is then defined from the maximum shoulder rotation till the ball release. Next, the `arm deceleration` phase is from the ball release to the maximum internal shoulder rotation. The final phase, `follow-through`, is from the maximum shoulder internal rotation until a balanced position is achieved. The text further explains that injuries are due to weaknesses in the kinetic chain of motion. This means

that an improper rehabilitation, which represents a weakness in a joint, can cause reinjury. These results are confirmed by [JTDJ09], which defines five parameters, which, if they are applied correctly, reduce humeral internal rotation torque, reduce elbow valgus load, and increase pitching efficiency. The parameters mentioned are:

- Leading with the hips
- Hand-on-top position
- Arm in throwing position
- Closed-shoulder position
- Stride foot toward home plate

All five of these parameters can be analyzed while the athlete is wearing the system. This will help physicians in classifying the risk of pitchers getting injured, while at the same time aiding in preventing injuries by helping them to improve techniques.

A recent article in the `New York Times` [Sch09] describes the rise of baseball injuries as a mystery. The article raises many different arguments that try to explain the mystery. The arguments are more or less convincing, but still don't explain the reason for all the injuries in a scientific manner. The last argument is the most convincing, as it says that as long as athletes are pushing themselves as hard as they can, the number of injuries will continue to grow higher and higher. For this reason it is necessary to assure that the athlete uses the right technique and to analyze the problems of a wrong technique.

A lecture about `Impingement` (muscles becoming irritated and inflamed) [NAD06] describes the difference between healthy and unhealthy pitchers. The differentiation is done by measuring the glenohumeral rotation by a goniometer. This procedure is normally only reserved for injured pitchers, and then only during a medical examination. An inertial measurement system together with a physical reconstruction of the movement can detect an abnormal glenohumeral rotation during physical exercise by comparing the left arm rotation with the right arm rotation.

The shoulder of an athlete is a complex statically and dynamically constrained system. The lecture of Brian D. Busconi [BDB06] describes shoulder instability as a reason for the higher risk of shoulder injuries. The slackness in a joint could be measured by the inertial measurement system, as it will result in an abnormally high acceleration due to the resulting impingement in the shoulder.

Michael Brown [MB06] describes the `mechanism of injury`. In his study, 66% of the total number of shoulder injuries are because of shoulder compression forces due to traction. These injuries are mainly from water skiing, grabbing overhead, from holding or pulling heavy objects or from throwing overhead. The forces creating these injuries can be derived from the accelerometers of the inertial measurement system.

1.3.2 Camera Systems

There are several camera-based optical tracking systems on the market. The two that appear to be industry standards are the XOS Technologies [XOS09] system and Motion analysis [Mot09] system. Both of these are passive camera-based systems based on marker tracking. The body of the athlete is equipped with several markers, which can be detected

and tracked by the camera system. This allows for accurate localization of the athlete's body segments.

A pilot study from the Harvard medical school [Ber07] shows that both of these systems have a drawback in giving just rough estimation of forces and torques on the body joints due to the second-order numerical differentiation. Their results showed that the forces and torques can be estimated with a precision of typically ±75% accuracy. These large errors are the result of the slow sampling rate of the camera system (400 Hz), which gives only about three data points at the peak of a pitch, as well as the position tracking, which needs a double numerical differentiation. The Inertial system measures accelerations and rates directly, therefore needs no numerical derivation. It samples with a much higher rate, therefore gets a much more precise estimation of the peak acceleration. In conclusion, camera systems are the best choice to make 3D visualizations, but they do not offer a reliable method of recording and calculating biomechanical data, especially where dynamic quantities such as forces and torques are involved.

Camera systems have also another drawback, as they are immobile and can exhibit problems with dynamic lighting conditions. Therefore they are not suitable for attempting to reconstruct the motions of players in a real game or in a realistic outdoor setting. For some players, it is important to have the pressure of the game on their shoulders to perform in their usual way. Furthermore, a motion tracking camera system is expensive and far from being easy to be installed. For the XOS system, for example, it is necessary to have each marker point seen by four different cameras for the whole measurement time to maintain a good accuracy. Our inertial measurement system can be typically applied to an athlete within five minutes, and can be used anyplace, opening the door to readily exploiting the system in realistic training scenarios.

1.3.3 Inertial Measurement Systems

Recently, several inertial measurement units have been developed in research or have come to the market. These are optimized for various purposes, but few of them with the intention of being applied in sports medicine. Nike has a commercialized product on the market [Nik09], which tags the distance the athlete was running as well as the calories the athlete has burned during the run. This permits to optimize the athletes training. Also a medical application presents [LA07], which describes a concept of an ear-worn sensor to detect injuries and gait impairment. Another system called body bug [Mya09] presents myapex. The system contains a three axes accelerometer, a temperature sensor and a skin resistivity sensor. The accelerometer detects motion and together with the temperature gradient and the skin resistivity it permits to count the calories within a 90% range.

A previous related project is the Sensemble [Ayl06], which used an ancestor of this inertial measurement unit. Sensemble was intended for interactive dance ensembles, but was also used in pilot studies with pitchers at Red Sox Spring Training in 2006. Sensemble was derived from an earlier project, the MIT Media Lab's Responsive Environments Group's Gait Shoe [Bam07] - a shoe designed for portable gait analysis, each Gait Shoe ported a wireless multimodal sensor node that (among other sensors) featured a three

axis gyroscope and three axis accelerometer. This project had its legacy in a pioneering instrumented shoe designed for interactive dance, the Responsive Environment Group's "Expressive Footwear" [JP00].

The group fielded also another related prior project, where an inertial measurement unit was used as a trainable user interface [Ben00, Ben07]. This system also used accelerometers and gyroscopes in order to recognize gestures. Interesting for this thesis is also the work of Vlasic, Popovic et al. [Dan07]. Their system combined accelerometer and gyroscope data with sonar time of flight data to determine location and orientation. The time of flight data is used to correct the drift of the inertial sensors, as it measures the distance between the sensors. The results of this work are positive, however the system is only sampled at 140 Hz and therefore is too slow for recording high speed activity. Furthermore, a backpack full of electronics is necessary for this system, and the sonar may introduce difficulty in this application (e.g., occultation by body segments during motion). A more recent relevant work is a wearable real-time system providing feedback during exercise [Bud08]. The system combines accelerometers and ultrasound to measure the arm position. The drawback is that this system is designed only to measure the arm position for the levels up, middle and down. Furthermore, 20 ms are necessary to record the position of the arm.

In [Kwa08], a wearable three axis accelerometer together with a linear encoder are used to get the joint angle. This method attains up to 98% of precision under laboratory settings by switching between the linear encoder and the accelerometer data.

Orient-2 [A.D06] is a system of wireless nodes having a three axis accelerometer, magnetometer and gyroscope. It is able to process sensor data with up to 15 nodes and 64 Hz in real time. This is achieved using a complementary quaternion-based filter and transferring the data in the form of a quaternion orientation. The paper [D. 03] presents a wired system using just accelerometers and magnetometers to do similar tracking of posture. As it is a wired system, it will be difficult to use it in high-speed athletic motion. Similarly, the IDEEA LifeGait [Min09] from MiniSun is a commercial system of wired accelerometers to determine gait and motion .

SHIMMER [Lor07] presents a wireless bluetooth platform with a three axis accelerometer. Its goal is to continuously monitor motor activity in rehabilitation patients. This system differs from others, in that it has a 2 GB microSD card, which stores the sensor data for up to 80 days. Its drawbacks are the 50 Hz sampling rate and that it is not clear how the synchronization is done.

Measureand [Mea09] presents two systems using fiber optic bend sensors, which measure joint angles. It works well for non athletic motion, but can be too bulky to be used for athletic activity, as it hinders natural motion.

Polhemus [Pol] and Ascension [Asc09] are also alternatives to cameras. Their 6 degree of freedom wireless magnetic sensor system allows up to 12 independent markers over large areas. The drawback of these systems is the usage of only magnetic sensors, which do not allow high sample rates and are therefore not well-suited to athletic application. Further the rates get more limited as more sensors are used in the system. These systems can be affected by the proximity of metal in the environment, and require setup and calibration each time they are installed.

Veltnik [DRPHV07] and Xsens [Xse09] present magnetic and inertial systems. These systems consist of a master control unit with a magnetic source and a system of nodes with 3 axis accelerometers, gyroscopes and magnetometers. In this way, the slow sampling rate of the magnetometer can be compensated by the data of the accelerometer and gyroscope. The drawbacks of the system are again the low sampling frequency (120 Hz maximum) as well as the belt-worn magnetic source, which hinders the athlete from his natural movement.

Biometrics [Bio09] is using another approach to measure joint angles. They are using electro-goniometers and get for one or two dimensions an accuracy of up to ± 2 degree. In [Koo04] these exoskeletons were used to analyze a golf swing. In [T.W80] the integration of subjective and objective data in the animation of human movement is analyzed using this method, which provides an impressive sampling rate, but the system also needs a master belt, which could disturb the sports activity. Additionally, it would be necessary to have an exoskeleton strapped at each side of the joint, which could also be restrictive.

As mentioned at the beginning of this section, this thesis is based on two previous masters theses from the Media Lab's Responsive Environments Group. The first master thesis from Ryan Aylward [Ayl06] was about dance and collective motion analysis. This system has been adapted for high-intensity athletic motion analysis by Michael Lapinski [Lap08], who transformed the system of Ryan Aylward to be usable for high-dynamic-range athletic motion. These changes include upgrading the microprocessor, the range of the sensors, and the range of the radio base station.

The results from his thesis show the accelerations and angular velocities, as well as the estimated velocities of the arm segments of a pitcher or batter. These quantities will also be used in this thesis to perform statistical analysis on the typical batting and pitching of individual players.

1.3.4 Inertial Sensors with Kalman Filtering

All these systems have in common, that they use the combination of accelerometer and magnetometer only to determine orientations of body segments. This means that the body segments are not driven by the accelerations measured by the accelerometers. Also, in [J.F06], abstract visual kinematic feedback coming from a video game does not give any real kinematic information about the forces that body segments and joints experience. The system combines the orientation of the earth acceleration and the heading of the magnetometer, which allows to determine the three dimensional orientation of segments which are not in motion. However the accelerations are not applied to the segments and therefore forces can not be determined. Further is the sampling frequency of 25 Hz not adequate for athletic motion tracking. Similar applications can be found in [You09], where the author compares Kalman filtering with complementary filtering in order to improve the quality of the data. A reduced-order Kalman filter concept is explained in [LEJP09], which uses the quaternion math to reduce the Kalman filter to the order of four.

Another quaternion-based complementary filter concept is presented in [ERB01]. Here the earth's magnetic field and the acceleration of gravity are used to remove the drift from

the gyroscope, as these two vectors allow to determine orientations in three dimensions.

In the paper [W. 08], a new algorithm is presented to filter the data of a two-axis accelerometer with an *Extended Kalman Filter* [MS01]. The idea of this system is to track the flexion angles. The results of this study indicate that the data processed using a Kalman filter are more precise than with a median filter.

A very interesting approach is also presented in the paper of J.A. Corrales et al. [JAC08], who used Kalman filtering to combine an inertial measurement system with an ultra wideband radio range finder. In his approach, the Kalman filter gets divided into two steps. One step to approximate the new position by the fast sampled inertial measurement system and the other step to correct the system drift by the ultrawideband localization system.

1.3.5 Inertial Sensors using Body-Models

The speckled golfer from [AB08] describes the first use of a full body 3D-motion tracking system Orient-2. It describes the motion of a golf swing and as a result it illustrates two rules, which permit to have a more efficient swing. In this paper the golfer-club system is modeled as a double pendulum to describe the body model.

An interesting approach is presented in [ZWS09] which used the Xsens [Xse09] sensor system combined with a physical body model. The idea was to measure human motion and classify physical gesture by using the data of an inertial sensor system and a 3D human body model. In this case, the body model is only used to correct the positioning of the segments, and not to calculate the forces and torques in the joints, which is a goal of this thesis.

2. The Concept of the Wireless IMU System

2.1 Introduction

Our IMU (inertial measurement unit) system is a wireless sensor network designed to track wide dynamic range movements. It is built for the purpose of measuring the accelerations and angular rates at different positions on the body of an athlete performing high performance athletic activity. Each identical node provides data at a high sampling rate, and the system as a whole is flexible enough to handle essentially any kind of activity. With a sampling rate of 1000 Hz (for the high range sensors) it provides a methodology for precise inference of velocities, forces and torques the human body experiences during physical activity. Applications other than sports medicine include, for example, health monitoring, immersive gaming, and interactive dance performance. The current system consists of five nodes, where each node measures linear accelerations and angular velocities along all three axes as well as measuring segment orientations with a digital compass/magnetometer. The wireless protocol running on the system is easily extensible, allowing for the inclusion of more nodes as needed.

2.2 Hardware Systems

2.2.1 The Global System

The hardware consists of two parts: The wearable battery-powered wireless nodes and a USB-powered basestation (figure 2.1).

Each node has a set of inertial sensors, a microcontroller, a wireless radio and a μSD

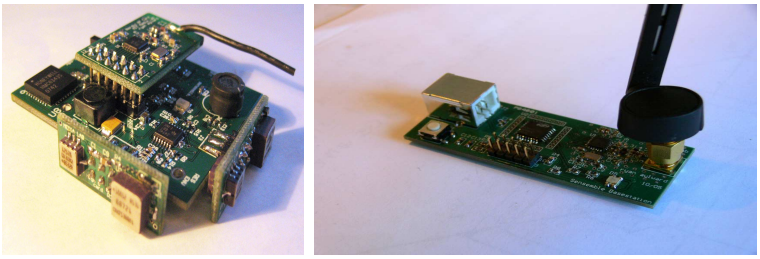

Figure 2.1: The Node (left) and Basestation (right)

memory card to save the data. The system does not aim to send the data in real time, but rather to store the data on the local μSD memory in order to avoid delays such as would be incurred as records are wirelessly uploaded (like in our earlier systems [Ayl06]), and thereby get the most data possible. Although data records can be wirelessly transferred upon request for diagnostic and verification purposes, the wireless network is primarily

used to synchronize the nodes and to indicate changes in the test subject (e.g., when new players are introduced or when a different kind of activity is begun) so the data can easily be segmented when the contents of the μSD cards are offloaded after a day of testing via a physical connection. As the radio is mounted on a daughter board, it can easily be upgraded as better radios are introduced.

The primary purpose of the basestation is for command and control of the network and synchronization of the nodes. It sends out commands to the nodes to tell them how to behave. Time synchronization between nodes is achieved by beaconing at 200 Hz with a time stamp included in the beacon packet. There are also commands to reconfigure the node, erase the flash memory and request a data packet from the node.

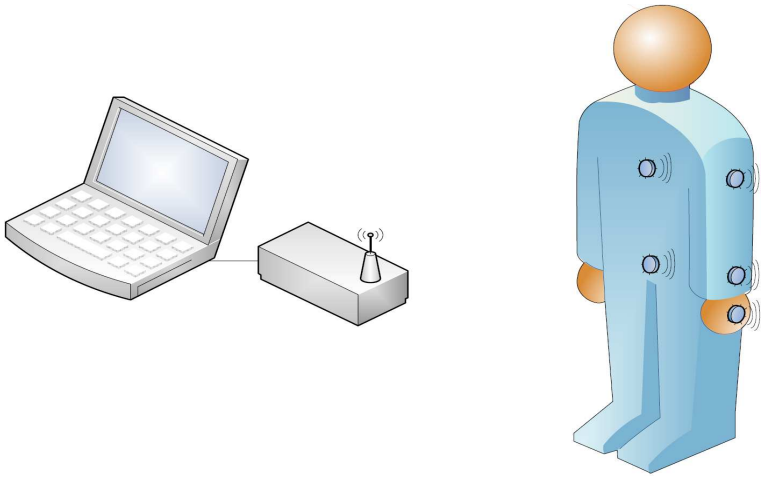

Figure 2.2: View of the Global System with a Person Wearing 5 Nodes

Figure 2.2 shows a typical application of the nodes on an athlete. One node is fixed on the hand, a second on the forearm, a third on the upper arm, a fourth on the sternum and the fifth will is placed at the waist. All these sensors are connected with the base station over a wireless 802.15.4 connection.

2.2.2 The Sensor Node

To measure high-speed athletic motion, it is not sufficient to use typical accelerometers. The accelerations measured on the arm of a pitcher are much higher than the range of typical accelerometers. A typical accelerometer has a range of up to $\pm 6\,\mathrm{g}$, which is sufficient for most movements, but not the movements of athletes. Therefore it is necessary to use special sensors for measuring high-G acceleration. These sensors are optimized for the detection of high-G acceleration (commonly used, for example, in impact detection), however their resolution for low-G movements is poor. Therefore, it is also necessary to use low range accelerometers to be able to precisely measure motion at low-G forces.

As in the case of the linear accelerations, the nodes are also equipped with gyroscopes that measure both high and low ranges. Typical gyroscopes with a resolution of $300\frac{\mathrm{deg}}{\mathrm{s}}$

have been augmented with special gyroscopes having a resolution of up to $10000\frac{\text{deg}}{\text{s}}$.
The system has a total size of $5,6\,\text{cm}$ by $5,1\,\text{cm}$ and weights 44 grams with battery. The

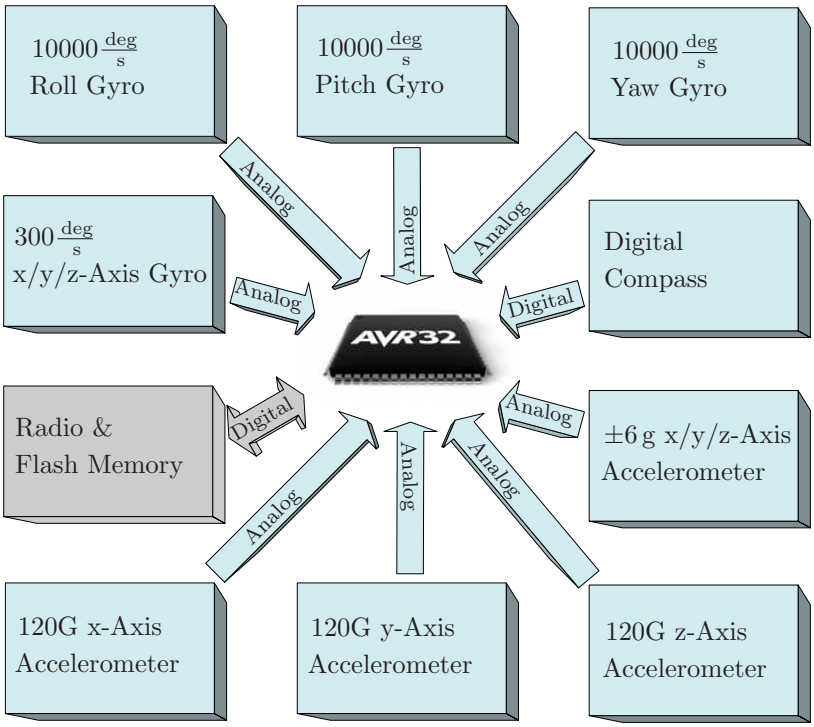

Figure 2.3: Sensor System of one Node

high-range accelerometers and gyroscopes, as well as the low range accelerometer are analog and are digitized by a 10-bit analog to digital converter. As it is necessary to sample with such a high rate, digital interfaces to sensors tend to be inadequate, because the communication ports on commonly available digital inertial sensors tend not to provide a high enough data rate to the microprocessor. For the low-range inertial sensors, a multiplexing was necessary, as there are not enough analog to digital converter ports available on our processor. The compass, on the other hand is digitally connected by a I2C interface to the micro controller. This compass has a low sample frequency (10 Hz) compared to the 1 kHz sample frequency of the high range gyroscope and accelerometer, but this is adequate to use in determining initial orientation and dynamically fusing the data together, for instance using a complementary filter [You09].

Gyroscope

The ADXRS300 [Inc09] from Analog devices and the IDG-300 from InvenSense [Inv] are $\pm 300\frac{\text{deg}}{\text{s}}$ yaw rate gyroscopes that are used for measuring slow angular movements with good precision. For the low range gyroscope, all three axes are now included in one package, so for this sensor it is no longer necessary to build orthogonal daughter boards. However the high range sensors are only available as single axis sensors, so orthogonal daughter boards

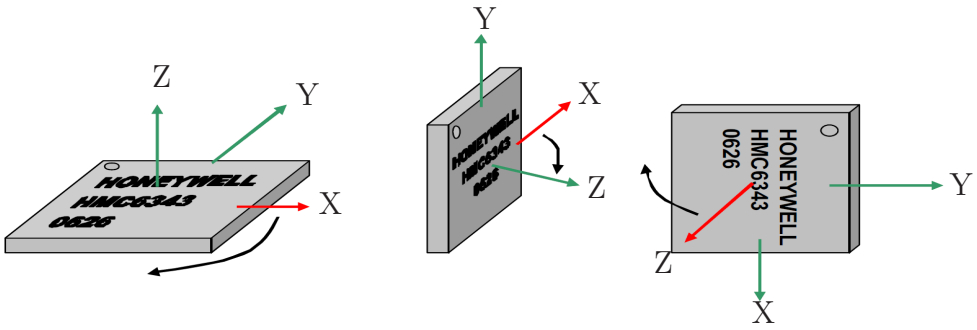

Figure 2.4: Compass Angular Measurements

are necessary (Figure 2.1). In addition to the low range measurement, the ADXRS300 has been modified to be able to measure high range rotations, up to $\pm 10000 \frac{\text{deg}}{\text{s}}$. The details of the application are written down in the application note [Har09] from Analog Devices.

High range Accelerometer

The ADXL193 [Inc10] from Analog Devices is used as high range accelerometer with a range of up to $\pm 250\,\text{g}$. In the node, only half of the maximum range is used ($\pm 120\,\text{g}$), as accelerations above $\pm 120\,\text{g}$ do not occur in our application. These sensors also measure only one axis, therefore two of the sensors have to be mounted on the orthogonal daughter boards.

Low range Accelerometer

As it can be seen from Michael Lapinski's master thesis [Lap08], to sensitively measure small accelerations, additional low range accelerometer are necessary. For this purpose, the accelerometer LIS344ALH [STM09] from ST-Microelectronics has been chosen. It is a three axis accelerometer with a range of up to $\pm 6\,\text{g}$. It has low power consumption and a small package size, which allows us to reduce the size of the PCB board. The output of all three axes is analog, which allows a fast sampling rate (333 Hz).

Digital Compass

The digital compass is used to get correct information about the heading, pitch, and roll angle of individual nodes. This is important information, as the gyroscope just gives an instantaneous angular velocity of the node. Compared with the gyroscope and the accelerometers, the compass delivers an absolute orientation.

The compass/magnetometer, an HMC6343 [Hon09] from Honeywell, has a duty cycle of only 6.67 Hz (every 150 ms), which is much slower than the 1000 Hz we have set for the high range sensors. It is connected over I2C with the micro controller to directly pass the data digitally.

Data Gathering Loop

A trade off has been made to reduce the amount of data produced every sampling cycle. The assumption is that the low range accelerometer and the low range gyroscope can be sampled just every 3 ms without losing precision. The idea is that the low range sensors are only relevant for slow movements, therefore it is not necessary to sample every millisecond for this data. In this way it is possible to divide the data for the low range sensors over three samples. Figure 2.5 shows the timing of the system. The high range data is presented

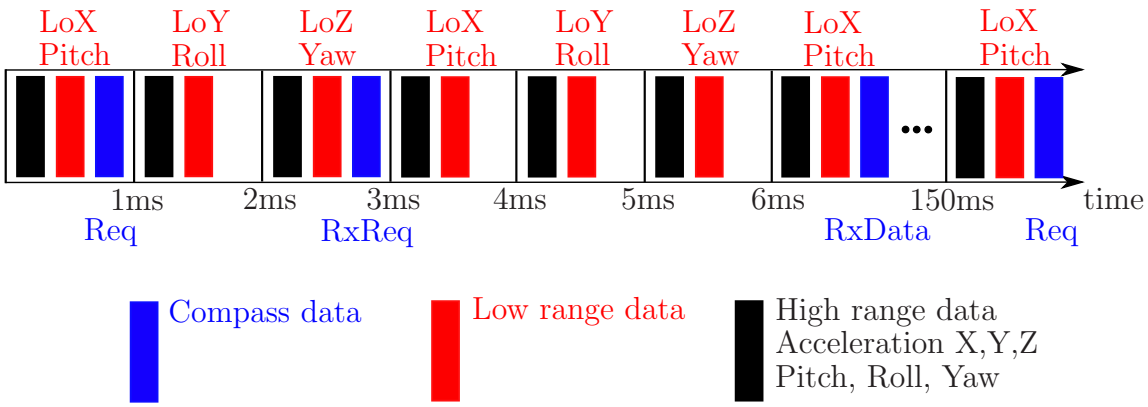

Figure 2.5: Timing

in black, the low range data in red and the compass data in blue. The colored rectangles represent the time slot it takes to acquire the high range, low range or compass data. For the high range data the accelerations and angular velocities of all three axis are sampled every millisecond. For the low range sensor data, only the linear acceleration and the angular velocity over one axis is sampled per time slot. The compass data is only requested every 150 ms (Req). The reception of the request is acknowledged two milliseconds later with RxReq and finally five milliseconds after the request the data reception is acknowledged with RxData.

2.2.3 802.15.4 Radio

Each node is equipped with a nRF2401a radio from Nordic Semiconductor [Nor09]. This low rate personal area network radio [Yan06] uses a 1 MHz channel spacing with a channel range in the industrial, scientific and medical (ISM) radio band from $2400 - 2527$ MHz and a maximum data rate of $1 \frac{\text{Mbit}}{\text{s}}$. Ideally the system would operate in real time, however, the amount of data produced by a group of nodes far exceeds the bandwidth of the radio. For the high range accelerometer and the high range gyroscope, for example, the amount

of data generated every millisecond is shown (Equation 2.1).

$$78bits/sample = \begin{cases} 10 \text{ bits High Pitch Gyro} \\ 10 \text{ bits High Roll Gyro} \\ 10 \text{ bits High Yaw Gyro} \\ 10 \text{ bits High X Accelerometer} \\ 10 \text{ bits High Y Accelerometer} \\ 10 \text{ bits High Z Accelerometer} \\ 16 \text{ bits timestamp} \\ 2 \text{ bits sample type} \end{cases} \tag{2.1}$$

In addition to this, the low range accelerometer (Equation 2.2), the low range gyroscope (which is sampled every 3 ms), and the compass data (Equation 2.3), which is sampled every 150 ms, has to be transferred.

$$78bits/sample = \begin{cases} 10 \text{ bits Low Pitch Gyro} \\ 10 \text{ bits Low Roll Gyro} \\ 10 \text{ bits Low Yaw Gyro} \\ 10 \text{ bits Low X Accelerometer} \\ 10 \text{ bits Low Y Accelerometer} \\ 10 \text{ bits Low Z Accelerometer} \\ 16 \text{ bits timestamp} \\ 2 \text{ bits sample type} \end{cases} \tag{2.2}$$

$$66bits/sample = \begin{cases} 16 \text{ bits heading} \\ 16 \text{ bits pitch} \\ 16 \text{ bits roll} \\ 16 \text{ bits timestamp} \\ 2 \text{ bits sample type} \end{cases} \tag{2.3}$$

Further, it is necessary to be able to use at least five nodes to track an athlete, which multiplies the necessary data to transfer by the number of nodes needed. Hence, you can see that the radio won't be capable of transferring the data in real time over the 1 Mbit channel - even with a zero bit error rate, which is far from what is realized in practice, especially with the player's body absorbing a considerable amount of the 2.4 GHz signal. Faster radios are available, but they are power hungry, which would result in necessitating a bigger battery and therefore more weight per node.

Now we can estimate how many sensor nodes could transfer the data in real time. The amount of bits per sample just has to be multiplied by the sampling rate to get the amount of data one node has to send out (Table 2.1).

high range accelerometer	$78\,\text{bit} \cdot 1000\frac{1}{\text{s}}$
high range gyroscope	$78\,\text{bit} \cdot 1000\frac{1}{\text{s}}$
low range accelerometer	$78\,\text{bit} \cdot 333\frac{1}{\text{s}}$
low range gyroscope	$78\,\text{bit} \cdot 333\frac{1}{\text{s}}$
compass	$66\,\text{bit} \cdot 6.7\frac{1}{\text{s}}$
total	$208384\frac{\text{bit}}{\text{s}}$

Table 2.1: Transfer rate necessary for one node

This result indicates that less than five nodes can be transferred in real time over the radio, as the radio can only transfer 1 Mbit and one node produces more than 200 kbit of data per second. The solution to this problem is to store the data locally on the node and transfer the data after the motion has been completed. In this way, the number of sensor nodes is not an issue any more, but the system loses the ability to track motion in real time. As there is no real time need for the analysis of our data, this solution is the most appropriate for this application.

2.2.4 Microcontroller

The microcontroller AVR32 UC3B from Atmel [Atm] is a 32-bit RISC processor running at 66 MHz. It is designed for high computation throughputs for deterministic and real-time control processes. It is been used because of its low power consumption. The core can be set by software into a power saving mode. It can handle the 1 kHz sampling rate together with flash memory writing. It provides a hardware I2C connection as well as SPI connection, and eight 10 bit analog to digital converter ports to digitize the values of the sensors.

2.2.5 Precision

A very important issue for this work is the precision of the inertial sensors, as they will be integrated over time and therefore could cause huge errors. In the following section the calibration of the sensors will be described. Here only the theoretical precision of the sensors will be estimated.

Low Range Accelerometer Precision

For the low range accelerometer, the range is from $-6\,\text{g}$ to $+6\,\text{g}$ over the range of 3.3 V sampled with a 10 bit analog to digital converter. Therefore, the theoretical precision of the sensor will be (Equation 2.4), not accounting for the problem of having temperature drift in the sensor value.

$$a_{\text{error}} = \frac{12\,\text{g}}{2^{10}} = 0.1149\frac{\text{m}}{\text{s}^2} = 0.0117\,\text{g} \tag{2.4}$$

This precision seems very high, but as it is necessary to integrate over time, this value can cause offset problems. In the following calculation, the error will be summarized over one second (Equation 2.5). The result of this integration is, as expected, a very large drift during this "long" time. "Long" here is used in the context of a single baseball pitch, which takes about 300 ms from start to finish.

$$\text{drift} = \int_0^T \int_0^T a_{\text{error}} \,\text{dt}\,\text{dt} = \frac{1}{2} a_{\text{error}} \cdot T^2 = 57.45\,\text{mm} \tag{2.5}$$

In order to propose a better resolution, it would be necessary to have an external analog-to-digital converter with 16 bits for the low range accelerometer and signal conditioning for the accelerometer to provide this level of signal-to-noise and work at $\pm 2\,\text{g}$. In this case, the precision with the reduced drift (Equation 2.6) will be:

$$\text{drift} = \int_0^{1\,\text{s}} \int_0^{1\,\text{s}} \frac{4\,\text{g}}{2^{16}} \frac{\text{m}}{\text{s}^2} \,\text{dt}\,\text{dt} = \int_0^{1\,\text{s}} \int_0^{1\,\text{s}} 5.9875 \cdot 10^{-4} \frac{\text{m}}{\text{s}^2} \,\text{dt}\,\text{dt} = 0.2994\,\text{mm} \tag{2.6}$$

Having this precision for the low range sensors would give a very interesting change in the resolution. This would be even hard to compete with high speed cameras giving out positions, as there will be a low drift in the accelerometers. However, it has to be considered, that there will be noise on the signal due to the hardware, so the precision would be less than the actual 16 bits from the analog to digital converter.

If the temperature of the accelerometer changes, this result has to be degraded, as there is a bias change of $0.4 \frac{\text{mg}}{\text{K}}$. This means that the acceleration value around zero will change by $0.0196 \frac{\text{m}}{\text{s}^2}$ for a temperature difference of 5 K. This is for example higher than the error we get from the digital to analog conversion for a 16 bit converter. Therefore the total change of the position would give:

$$\text{drift} = \int_0^{1\,\text{s}} \int_0^{1\,\text{s}} (0.01962_{\Delta 5\,\text{K}} + 5.9875_{\text{Sample}} \cdot 10^{-4}) \frac{m}{s^2} \,\text{dt}\,\text{dt} = 10.11\,\text{mm} \tag{2.7}$$

High Range Accelerometer Precision

For the high range accelerometer, the resolution is naturally worse than the low range precision, but the time over which the high range accelerometer is needed is much smaller than the time during which the low range accelerometer is needed. In the experiment, the high range data is only needed for about 20 ms. This gives a drift of 2.8:

$$\text{drift} = \int_0^{20\,\text{ms}} \int_0^{20\,\text{ms}} \frac{240\,\text{g}}{2^{10}} \frac{\text{m}}{\text{s}^2} \,\text{dt}\,\text{dt} = 0.9197\,\text{mm} \tag{2.8}$$

Also, in this case the analog to digital converter could (theoretically) be exchanged for a 16 bit converter, but as the drift is already below one millimeter, there should be no need for it.

Low Range Gyroscope Precision

For the low range gyroscope, the same problem applies. Here we measure an angular velocity, so there would not be a problem with a quadratic error as with the accelerometer, since there is only one integration to obtain angular displacement. The following Equation 2.9 gives the output error due to the analog-to-digital sampling.

$$\theta = \int_0^T \frac{300}{2^{10}} \frac{\text{deg}}{\text{s}} \,\text{dt} = \frac{300}{2^{10}} \frac{1}{\text{s}} \cdot 1\,\text{s} = 0.2930\,\text{deg} \tag{2.9}$$

This error is quite small and should therefore be no issue for the simulation. An additional oscillation on the signal creates an error of $\pm 0.05\,\text{V}$ on the output [Inc09], but this value is below the error we get from the analog to digital conversion.

High Range Gyroscope Precision

For the high range gyroscope the same thought can be used as with the high range accelerometer. Also this sensor data will only be used during about 20 ms. The following equation (Equation 2.10) shows the approximative error value.

$$\theta = \int_0^T \frac{12000}{2^{10}} \frac{1}{\text{s}} \,\text{dt} = \frac{300}{2^{10}} \frac{1}{\text{s}} \cdot 20\,\text{ms} = 0.2343\,\text{deg} \tag{2.10}$$

Conclusion

It makes only sense to use a better (16 bit) analog to digital converter for the low range accelerometer, as the error due to the drift is not tolerable. For other sensors, a better analog to digital converter would not affect the error due to other factors, or the error is already small enough and does not need to be further corrected.

16 bit sampling can be done by an external low power analog to digital converter, which sends out the converted signal over a PWM signal to the micro controller. Therefore, only one digital input of the micro controller will be needed. But that said, the electronics design needs to be able to provide enough signal-to-noise to enable 16 bits worth of dynamic range.

The momentarily hardware has some spikes on the analog signal due to the 33 Hz LED pulsing on the board. Without having the LED on the board, these spikes decrease by at least a factor of two. This result can be even improved by not pulsing the LED at all. Furthermore the spikes can be reduced by using a longer integration time for the analog to digital sampling. Another issue of the regular hardware is that the analog and digital ground is not separated properly. Therefore all digital switching signals interfere within the analog signals.

2.3 Calibration

A calibration of the individual sensors is necessary to obtain reliable physical quantities. Calibration ensures that the node's output accurately reflects physical values. The calibration of the system should be done before the measurements are taken, as it can't be done if a given node gets damaged during use.

In theory, every physical value can be obtained from a conversion factor given by the manufacturers' datasheets. In practice, these numbers are not highly accurate because of different production conditions for every single chip. These issues can generally be approximated by a normal distribution, but nevertheless some components may differ more than others. Therefore, it is necessary to run a calibration for every single sensor on every single axis.

For gyroscopes, the best way to calibrate is to use a rotating device with a known angular velocity. Using different angular velocities and relating them to the output voltage of the gyroscope gives us a good calibration for the sensor. It has to be remarked at this point that the high range gyroscopes are externally biased to obtain their higher range [Inc10].

The accelerometer needs to be moved with a known acceleration, and the data has to be gathered to get the calibration done. However in this case literal calibration is much more complicated, because it is difficult to have a linear acceleration of around 100 g for a given period of time [Lap08]. Here, centripetal acceleration from rotation can be applied.

2.3.1 Calibration with an electric drill

As described above, it is necessary to spin the gyroscope with a fixed velocity around the pitch, roll and yaw axes. To generate this spin, a common variable speed electric drill is used, which can turn at $\pm 2500\,\mathrm{RPM}$. To ensure that the rotation speed of the drilling machine is correct, a digital tachometer is used. To fix the node properly into the drilling machine, custom bracket nodes were designed using Solidworks. The laboratory's internal 3D printer was used to fabricate the part. The problem was then to resist damage from the high centrifugal forces from the high speed rotation of the drilling machine [Lap08].

2.3.2 Gyroscope Calibration

Because of a different bias of every gyroscope, the calibration had to be done with every node. Six different velocities for the calibration have been used:

- $\pm 10.000 \frac{\mathrm{deg}}{\mathrm{s}} - 1,667\,\mathrm{RPM}$
- $\pm 9.000 \frac{\mathrm{deg}}{\mathrm{s}} - 1500\,\mathrm{RPM}$
- $\pm 6.000 \frac{\mathrm{deg}}{\mathrm{s}} - 1000\,\mathrm{RPM}$
- $\pm 3.000 \frac{\mathrm{deg}}{\mathrm{s}} - 500\,\mathrm{RPM}$
- $\pm 1.000 \frac{\mathrm{deg}}{\mathrm{s}} - 167\,\mathrm{RPM}$
- $0 \frac{\mathrm{deg}}{\mathrm{s}} - 0\,\mathrm{RPM}$

The average of an eight second dataset is used. This average creates a line, which gives us the slope and offset of this individual gyroscope.

2.3.3 Accelerometer Calibration

For the accelerometer calibration the drill is also used. Here the sensing axis of the accelerometer is orthogonal to the center of rotation in order to create a centripetal acceleration. To calculate the acceleration from the rotational velocity the elementary physics of the centripetal forces can be used [Ric73].

$$\omega_{\mathrm{RPM}} = \left(\frac{30}{\pi}\right)\sqrt{\frac{9.81 * G}{r}} \tag{2.11}$$

ω_{RPM}	Angular velocity in rounds per minute
G	Number of G's (earth acceleration)
r	Radius from the center of mass

To obtain the desired acceleration of 100 g, it is necessary to have enough lever arm (distance between the node and the center of rotation).

2.3.4 Offset Recalibration

The calibration described above results in a distance from zero of more than 20 g in some cases, therefore it is still not possible to use the data as it is. The offset of a calibration is the distance to zero when no movement occurs.

Therefore, an algorithm has been developed to remove the offset, or at least to reduce it to a minimum. It is not sufficient to assume that taking the mean over a long period of time will give a close approximation of no motion.

The idea is to detect the data points where the person is not moving during the testing, and set them to zero. The problem is that these points are not at zero when the mean value over the total amount of data is calculated. It is easy to detect this flat line with the eye, however it is difficult to compute it (Figure 2.6). Therefore the algorithm to detect this flat line has been split into two parts. The problem is if the mean value is built using values of the fast movement. The first pass of our solution is to compute the mean value of the data - a second pass takes only the points around this mean value to compute a more realistic mean value. The only values that are relevant are the values where the node is not moving much in order to keep the error small. Therefore, the algorithm first builds a mean value over the whole data vector, than fixes a small boundary around the mean value to remove the fast movements and finally computes the mean value over the data that stays in the boundary. This algorithm is then iterated using increasingly smaller boundaries to step closer to the actual zero offset point.

The result gives an almost perfect set of data with no offset. Figure 2.7 shows that the flat line of the y-axis is set to zero using the algorithm described in this section. It shows also the boundaries of the algorithm around the mean value, which get increasingly narrow with every step.

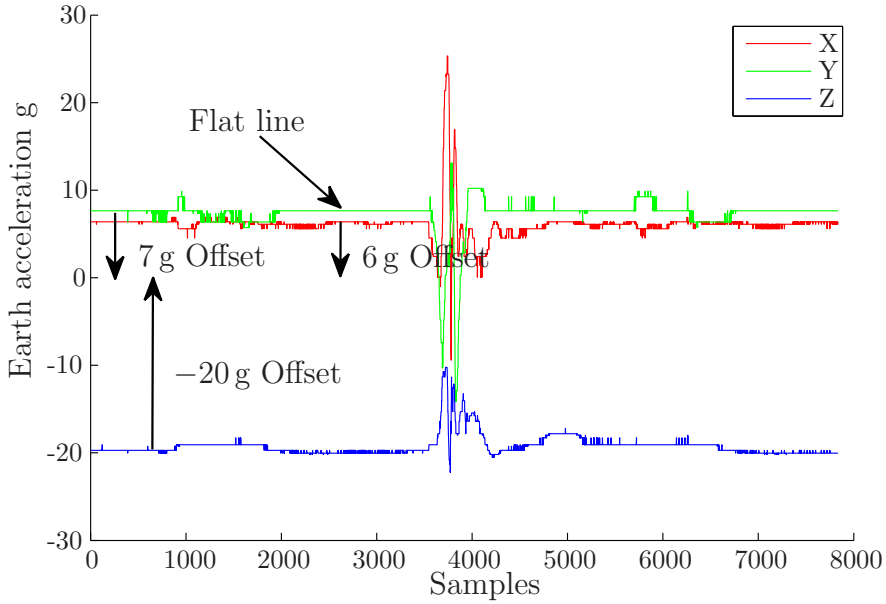

Figure 2.6: Detection of a Flat Line

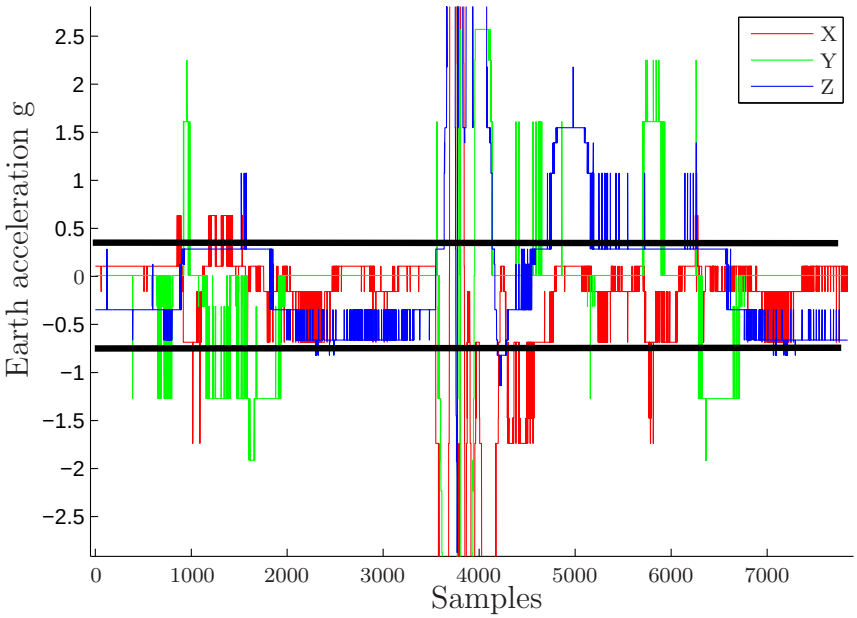

Figure 2.7: Flatline set to Zero

2.3.5 Accelerometer Recalibration

Due to previous errors in earlier calibrations with the electric drill, data from the low range accelerometer was not usable. As the calibration with the electric drill was not only erroneous caused by unbalanced spinning and changing spin velocity , but also time consuming another method will be presented here. This method is quite simple and precise in the same time. The Idea is to use the Earth's acceleration as a reference point. By placing the sensor orthogonal to the ground three data points can be gathered, one for positive earth acceleration (+1 g), one for negative earth's acceleration (−1 g) and one for no earth acceleration (0 g). With a precision rotating fixture, this could even done for a rotation of 45 deg , which would give two more points. With this simple calibration only three points are used with a least square curve fitting method, resulting in a repeatable, well-fit line through these points.

Our sensor system gets values for the low range sensors for eight seconds in 3 ms time increments. Therefore it is enough to repeat the three positions each three times. To increase the precision of this method, the data of the sensor has been analyzed for each position with the method described in Subsection 2.3.4. In this way, the sigma of the approximation got much better, or even approached zero (Table 2.3), as the new vector did not even change its digitalized value.

Axis	**+1 g**	**0 g**	**−1 g**
X-Axis	448	512	575
Y-Axis	575	508	440
Z-Axis	448	508	575

Table 2.2: Offset of the Accelerometer Recalibration (raw digital value)

The offset of the accelerometer can now easily be calculated by extrapolating our fit line to zero acceleration. In the following Table 2.2, the three points with offset are displayed. The offset is in the middle of the 10 bits and therefore settles around 512. This allows to, measure in the positive as well as negative direction. These values give the slope and

Axis	**+1 g**	**0 g**	**−1 g**
X-Axis	0	0.4742	0.0113
Y-Axis	0	0.0920	0
Z-Axis	0	0.8154	0.0478

Table 2.3: Sigma of the Accelerometer Recalibration

the offset of the curves approximated by least-square fitting. Knowing that the curve of these accelerometers is a straight line, it is adequate to use a first order polynomial. The result of this approximation can be seen in Figure 2.8. Here it has to be noted, that one of the axes has a negative slope. This is because the y-axis has been defined in the opposite direction to match up with the y-axis of the other sensors.

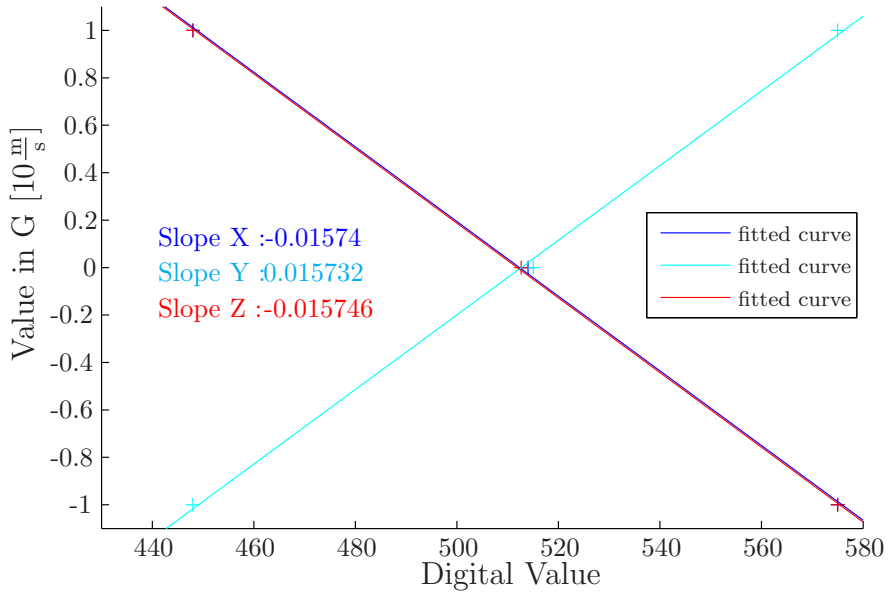

Figure 2.8: Low Range Acclerometer Slope

In order to see the error of this assumption, a Gaussian approximation has been taken and compared to the ksdensity function [MAT10], which computes a probability density estimate of the sample vector (Figure 2.9). From this comparison, it can be seen that the Gaussian approximation is correct and precise enough for the matter of calibration. To get a Gaussian distribution that totally fits with the Ksdensity function it is necessary to modify σ by a factor of 1.7, but the expected value is the same. Therefore we can say that the approximation with an ideal Gaussian function is precise enough to get the expected value. The ideal Gaussian distribution just makes a little error on the variance, however there is no error on the approximation of the process to be a Gaussian distributed process. This comparison is mentioned here to prove that the distribution is Gaussian and therefore modeled by an ideal Gaussian distribution.

2.3.6 Low Range Gyroscope Recalibration

As already described in the low range accelerometer recalibration, the electric drill does not have a stable enough angular velocity at low speeds to allow a good calibration. To get a much better result, a turntable (Figure 2.10) is used, which provides an almost perfect stable angular velocity and almost no vibration. These conditions are ideal to allow a good calibration. To verify the rotation speed, ω, of the turntable, a digital tachometer is used. To get the real angular velocity the measured speed at the tachometer is divided by the number of the reflective stripes on the platter (Equation 2.12). Here four stripes are used because the rotation is quite slow and more stripes are used to improve the resolution

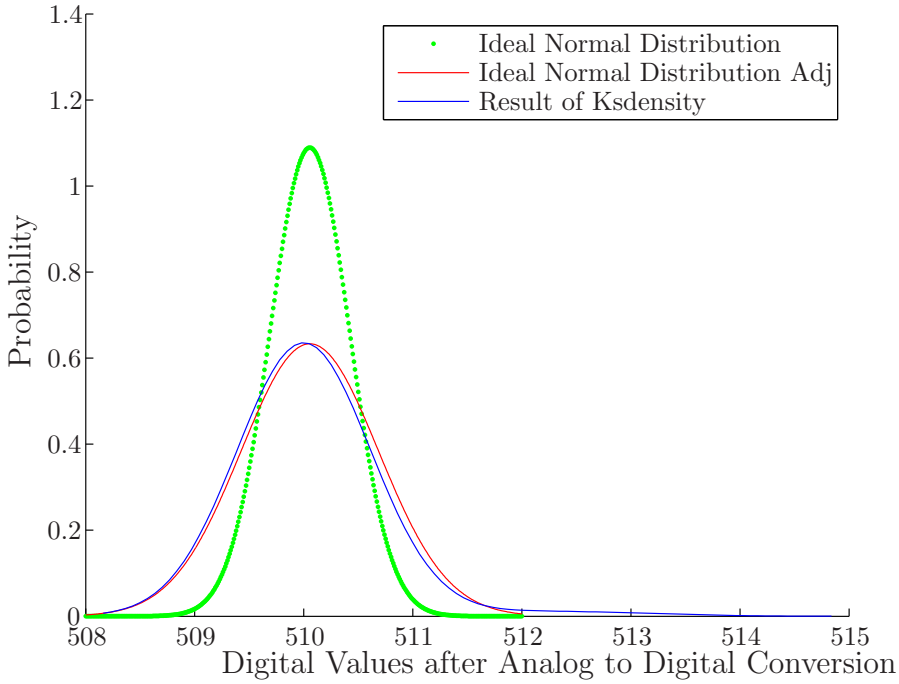

Figure 2.9: Comparison of Gaussian and Ksdensity

(Equation 2.12).

$$\omega_r = \frac{\omega_T}{N_{stripes}} \tag{2.12}$$

ω_r	Real angular velocity
ω_T	Velocity of the tachometer
$N_{stripes}$	Number of stripes

The result of a first order approximation of the five different angular velocities can be seen in Figure 2.11. As the maximum angular velocity of the low range gyroscope lies at about $300\frac{\text{deg}}{\text{s}}$, the upper limit point was chosen to get the most precision possible. The other value is at around $200\frac{\text{deg}}{\text{s}}$, which is the upper third of the range. The same points are used in the negative direction to get a symmetric distribution of points around the x-axis. In summary, we can say that the calibration is precise to at least 0.2%.

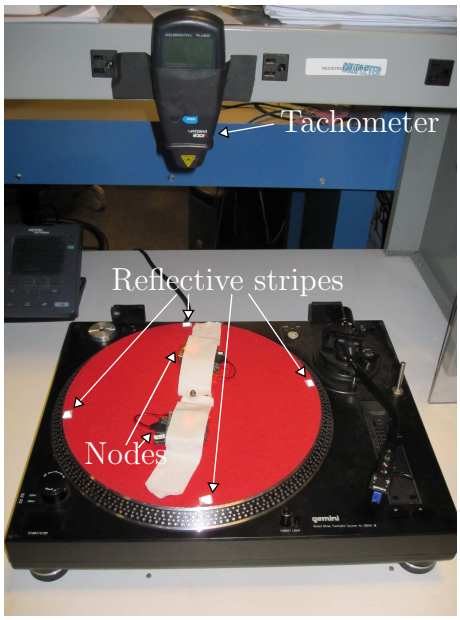

Figure 2.10: Gyroscope Calibration Using a Turntable

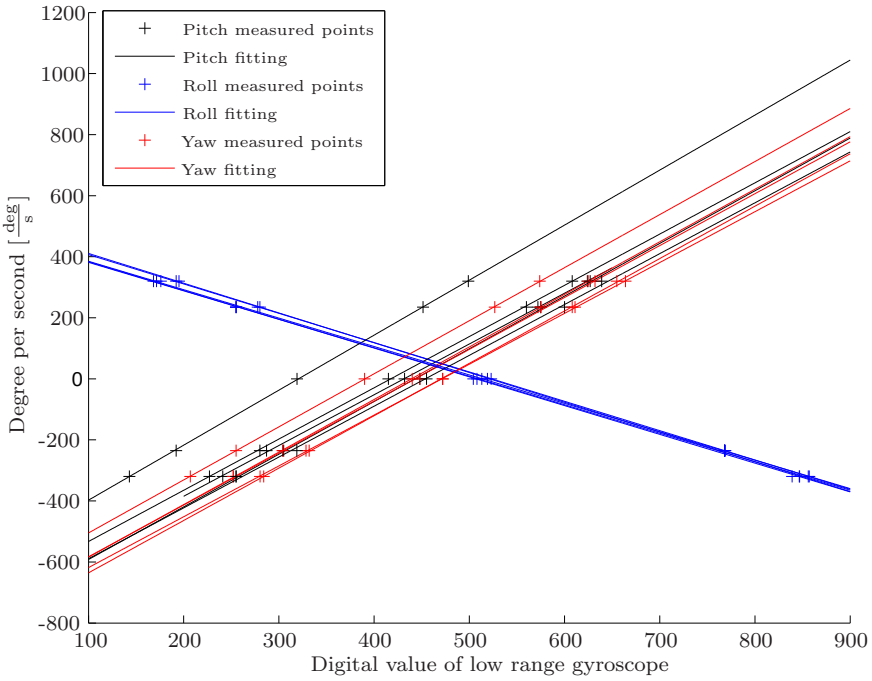

Figure 2.11: Calibration Results of the Low Range Gyroscope

3. The Management of Data

Every node, while it is sampling, produces a huge amount of data (about 1669 data points per second), which need to be stored somewhere in order to perform analysis. As it is quite difficult to store data in files and to manage all of these files, a database schema has been developed. To handle loading and reading data from the database, a user interface and data processing engine has been conceived, which allows access and modification of the data and allows users to view data in various forms.

3.1 Software Structure

C++ is used for the calculation of the inverse kinematics and Visual Basic for the user interface and analysis tool. The Visual Basic programming language is used for the realization of the user interface and to analyze the data. The C++ inverse kinematics program, which calculates the forces and torques at the joints of the bones, can be included in the visual basic user interface using a dynamic link library. In order to keep the program as fast as possible, the C++ program connects directly to the database and does not query the visual basic user interface. The following diagram shows the relationships between the high level layers (Figure 3.1).

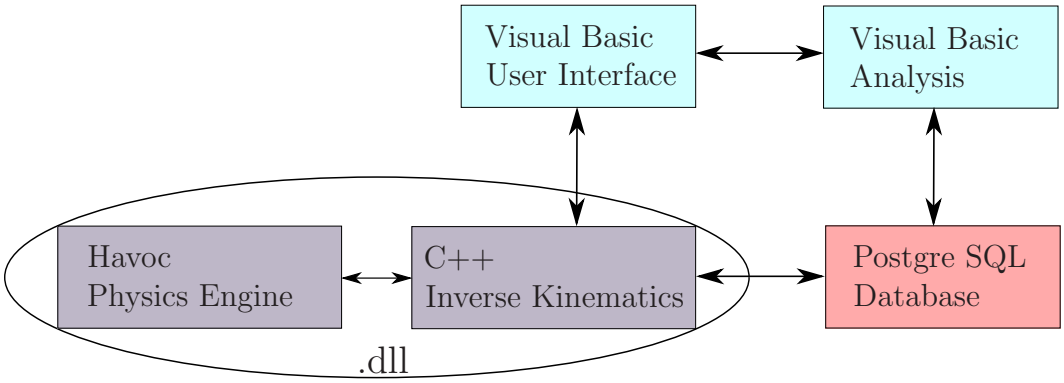

Figure 3.1: Software Structure

The C++ inverse kinematics program uses the Havok [cCTRLtH09] physics engine, which is used for scientific research and gaming applications. The relational database system used is postgres SQL and was chosen because Postgres is optimized for tables with large sets of rows. It is accessed over the network and allows us to have one global database to collect all the data.

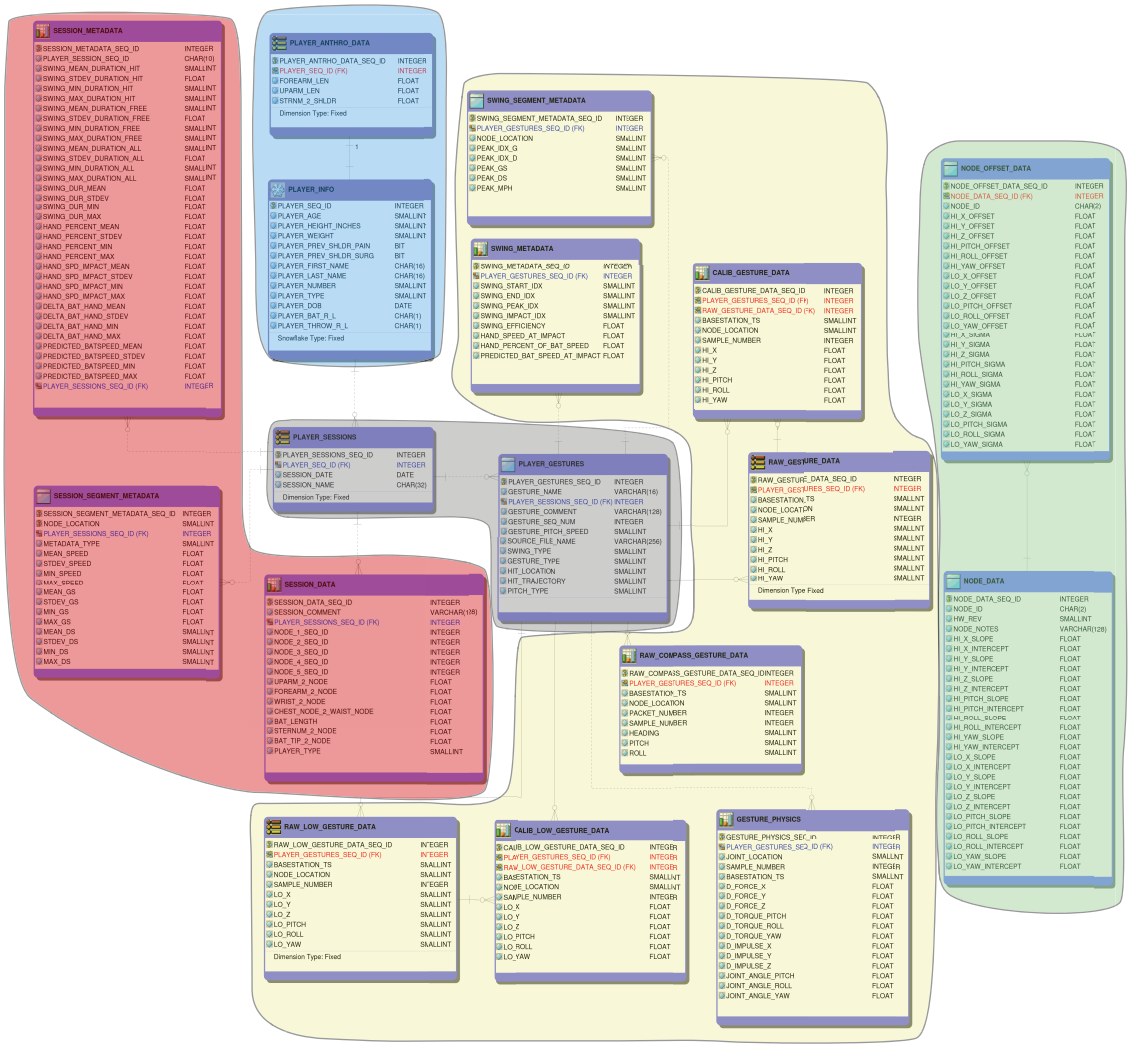

Figure 3.2: Database Structure: Session, Player, Gesture, Node, Interface

3.2 Database Structure

The database consists of five main parts, which are enlightened in color in Figure 3.2. There is the player information, such as weight, age, previous injuries and anthropometric data, the sensor node data, where all the slopes and offsets of the sensors are stored, and the gesture data, which stores the accelerations, angles and angular velocities for every time point and also some statistical data for every gesture. Out of the physical model, the forces, torques and joint angles are stored for every gesture at every joint. Finally, the session information stores information about the position of the sensors during the training session and the statistical analysis data over all gestures. This database allows a user to access the data from any computer and allows parallel access from different users. The

structure of the database allows the user to build any kind of statistics from the data. At the same time, it is easy to change the number of nodes used. Only one table has to be changed in this case which can be implemented very quickly.

3.2.1 Player Information

The table for the player information (*PLAYER_INFO*) contains biographical information about a player:

- Player Sequence ID - Number that rises for every player entry
- Age
- Height
- Weight
- Previous shoulder pain
- Previous shoulder surgery
- First name
- Last name
- Number
- Type - Batter or Pitcher
- Date of birth
- Batting hand - right or left
- Pitching hand - right or left

This table has a one to one relation with the player anthropometric data. The table *PLAYER_ANTHRO_DATA* stores:

- Anthropometric Sequence ID - Number that rises for every anthropometric entry
- Player Sequence ID - Number that relates the player and the anthropometric data
- Forearm length
- Upper arm length
- Sternum to shoulder length

This information helps to reconstruct the player in the virtual space. Necessary information that is not stored here is that of the individual segment mass. This data is available from [Win05], and is based on percentages of the player's entire weight. For example the upper arm weight is about 2.8% of the entire body weight. The book has tables that represent the weight distribution of a human body over all segments. Therefore, even without the knowledge of the weight of each segment, the weight of each segment can be approximated from the total weight of the body. Hence it is possible for us to construct a body model with each segment having an estimated weight.

3.2.2 Gesture

The gesture data is grouped in the *PLAYER_GESTURES* table, which represents information about the type of swing or pitch and gives every gesture an individual identification number. This number called *PLAYER_GESTURES_SEQ_ID* is then used to identify the raw and calibrated data of the gesture in the tables:

- *RAW_COMPASS_GESTURE_DATA*
- *RAW_LOW_GESTURE_DATA*
- *CALIB_LOW_GESTURE_DATA*
- *RAW_GESTURE_DATA*
- *CALIB_GESTURE_DATA*
- *SWING_METADATA*
- *GESTURE_PHYSICS*

The *RAW_GESTURE_DATA* table contains sampled data of the high range accelerometer and high range gyroscope and *RAW_LOW_GESTURE_DATA* the low range accelerometer as well as the low range gyroscope data in three dimensions. The *CALIB_LOW_GESTURE_DATA* and *CALIB_GESTURE_DATA* table contains the same information but modified by the information from the *NODE_DATA* and the *NODE_OFFSET_DATA* applying the offset and the slope of each sensor. The information stored in these tables is for each table:

- Gesture sequence ID - Relates the accelerations and angular velocities to a gesture
- Acceleration in x axis
- Acceleration in y axis
- Acceleration in z axis
- Angular velocity around the x axis (pitch)
- Angular velocity around the y axis (roll)
- Angular velocity around the z axis (yaw)

As the compass is already internally calibrated, there is no table containing the calibrated data of the compass, but only the raw data of the compass. As the compass is not sampled as fast as the other sensors, the table has also an entry called sample number, which helps to relate the compass data with the data of the other sensors. In the future one additional table will be created, which stores the magnetometer data. The magnetometer data allows to use self conceived algorithms to compute the orientation and therefore it could correct the result of a not properly calibrated compass orientation.

Gesture Statistics

The table *SWING_METADATA* contains statistics about the gesture:

- Swing metadata sequence ID - Number that rises for every statistics table added
- Gesture sequence ID - Relates the statistics to a gesture
- Swing start index
- Swing end index
- Swing peak index
- Swing impact index
- Swing efficiency
- Hand speed at impact
- Hand percent of bat speed
- Predicted bat speed at impact

This information together with the data in the *SWING_SEGMENT_METADATA* we are able to build the session statistics.

- Swing segment metadata sequence ID - Number that rises for every new entry
- Gesture sequence ID - Relates the statistics to a gesture
- Node location
- Peak index acceleration
- Peak index angular velocity
- Peak acceleration
- Peak angular velocity
- Peak speed in mph

Gesture Physics

The last table of gesture is the table *GESTURE_PHYSICS*. This table saves the forces and torques calculated for each joint from the physical simulation:

- Gesture physics sequence ID - Number that rises for every new entry
- Player gestures sequence ID - Relates the physics to a gesture
- Joint location
- Sample number
- Base station
- Force in x axis
- Force in y axis
- Force in z axis
- Torque around the x axis (pitch)
- Torque around the y axis (roll)
- Torque around the z axis (yaw)
- Impulse in x axis
- Impulse in y axis
- Impulse in z axis
- Joint angle around the x axis (pitch)
- Joint angle around the y axis (roll)
- Joint angle around the z axis (yaw)

At the moment, these values represent the most relevant information to our medical researchers. From a medical point of view, we don't just need to know the angle through which the hinge is able to be moved, but rather the forces that are acting to move it in the direction where it can't be moved.

3.2.3 Session

A session contains a number of gestures and allows a statistical overview on the gestures, as well as some informations about the placement of the nodes during the training. It consists of three tables. The *SESSION_METADATA* saves the global statistics over the whole amount of gestures in the session:

- Session metadata sequence ID - Number that rises for every new entry
- Player session sequence ID - Relates the metadata to a session of a player
- Swing mean duration of a hit
- Swing standard deviation of the duration of a hit
- Swing minimum duration of a hit
- Swing maximum duration of a hit
- Free swing mean duration
- Free swing standard deviation of the duration
- Free swing minimum duration
- Free swing maximum duration
- All swings mean duration
- All swings standard deviation of the duration
- All swings minimum duration
- All swings maximum duration
- Hand mean speed in the percentage to bat speed
- Hand standard deviation speed in the percentage to bat speed
- Hand minimum speed in the percentage to bat speed
- Hand maximum speed in the percentage to bat speed
- Hand mean speed at impact
- Hand standard deviation speed at impact
- Hand minimum speed at impact
- Hand maximum speed at impact
- Mean time difference between bat and hand maximum
- Standard deviation time difference between bat and hand maximum
- Minimum time difference between bat and hand maximum
- Maximum time difference between bat and hand maximum
- Predicted bat mean speed
- Predicted bat standard deviation speed
- Predicted bat minimum speed
- Predicted bat maximum speed

This information is important as it gives the ability to statistically distinguish the different types of players. After a long term analysis it could be even used to evaluate the swing of a player and use it to choose the most prestigious young player for the next season. The table *SESSION_SEGMENT_METADATA* stores all the statistical data of one segment in the session:

- Session segment metadata sequence ID - Number that rises for every new entry
- Node location
- Player session sequence ID - Relates the segment metadata to a session of a player
- Type of segment - Hand, Forearm, Upper arm, Chest and Hip
- Mean speed
- Standard deviation of the speed
- Minimum speed
- Maximum speed

- Mean acceleration
- Standard deviation of the acceleration
- Minimum acceleration
- Maximum acceleration
- Mean angular velocity
- Standard deviation of the angular velocity
- Minimum angular velocity
- Maximum angular velocity

The third table in the session is the table *SESSION_DATA*. This table stores all the information about which node categories have been used and where they are placed on the player:

- Session data sequence ID - Number that rises for every new entry
- Session comment
- Player sessions sequence ID - Relates the session data to a session of a player
- Hand node sequence id (node 1)
- Forearm node sequence id (node 2)
- Upper arm node sequence id (node 3)
- Chest node sequence id (node 4)
- Waist node sequence id (node 5)
- Distance from upper arm to node
- Distance from forearm to node
- Distance from wrist to node
- Distance from chest node to waist node
- Bat length
- Distance from sternum to node
- Distance from bat tip to node
- Player type - pitcher or batter

The node placement is different for every session, as the nodes are never exactly in the same place. The session is not directly related to a certain player, because the physical shape of every player can vary between the different sessions (e.g., a player's muscle distribution changes over training).

3.2.4 Node Data

The sensor node information is grouped in two tables. The first table *NODE_DATA* contains the slope of each sensor on the node and the y intercept, which is used to remove offset (Section 2.3). This is stored for the low range (LO) and the high range sensors (HI). In addition to this information, a column listing the hardware revision (HW_REV) and a column for notes (NODE_NOTES) is added. The second table, *NODE_OFFSET_DATA*, contains the offset of each sensor on the node (OFFSET) and the statistical deviation of each offset (SIGMA) to be able to predict the error which has occurred during the calibration.

3.2.5 Table Relations

Player Session Relation

Two tables are used to relate the player, session, and gesture information together. The table *PLAYER_SESSIONS* relates the identification number of the session (PLAYER_SESSIONS_SEQ_ID) to the identification number of the player (PLAYER_SEQ_ID). The following two columns store the session date and the session name the user want to associate with the identification number. This information might be interesting when the database has a huge amount of data, e.g., to find a certain session in the database.

Player Gestures Relation

The *PLAYER_GESTURES* table relates the gesture (PLAYER_GESTURES_SEQ_ID) and session sequence identification number ((PLAYER_SESSIONS_SEQ_ID)). By these two identification numbers, the table relates the gesture to a player. Furthermore, the gesture can have a specific name (gesture name) to enable a user to find it again when a huge amount of data is stored in the database. This table has also a column reserved for comments (gesture comment), which allows a user to write down some interesting points concerning this gesture. The other columns of the table are:

- Gesture sequence number - a number that increments for every new entry
- Gesture pitch speed
- Source file name - name of the file imported in the database
- Swing type - free swing, hit, strike, ...
- Gesture type - pitch, bat, ...
- Hit location
- Hit trajectory
- Pitch type - fast ball, slow ball, ...

Therefore this table is not only a relation table between the parts of the database, but also stores some general information about the gesture.

3.3 User Interface

The user interface is written in Visual Basic for ease of programming and because it allows integration of a C++ program by using a dynamic-link library (dll). The interface is built for end users to access, update and import data into the database.

The interface is grouped the same way as the database. The letters A to H in Figure 3.3 mark the different tabs the user can open up. The tab marked with A is the player information. Here the player anthropometric data (table *PLAYER_ANTHRO_DATA*) as well as the player information (*PLAYER_INFO*) can be accessed, updated or included.

At the tab marked with a B the session of the player can be accessed. Here all the information of the table *SESSION_DATA* can be accessed.

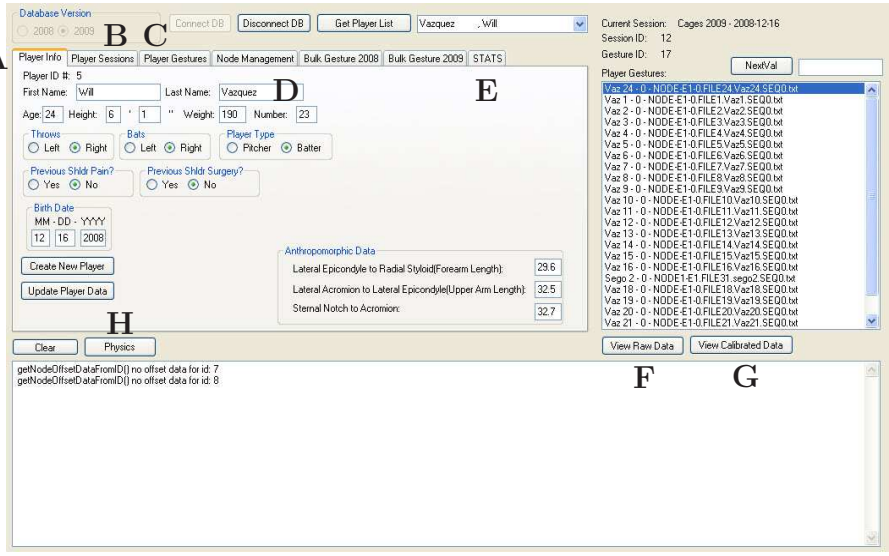

Figure 3.3: User Interface Main Frame

Under the letter C the player gesture, which means basically all the tables in the database related to the gesture, can be accessed.

Under D the node information (*NODE_DATA* and *NODE_OFFSET_DATA*) can be accessed. The following two tabs are used to upload the accelerometer and gyroscope data in the database, but do not give access to the database.

The last tab, E, provides access to the statistics of the gesture (*SWING_SEGMENT_METADATA* and *SWING_METADATA*), as well as to the statistics of the session (*SESSION_SEGMENT_METADATA* and *SESSION_METADATA*).

As you can see, the user is able to access basically every part of the database. The parts are grouped by different tabs and give access to all information stored in the database. The GUI also allows users to update data in the database, if some information needs to be modified.

The two buttons marked with F and G give the user the ability to plot the accelerometer, gyroscope and compass data over time for either the raw data or the calibrated data.

The button H allows the user to start the physical engine to calculate the forces and torques in the joints. In this case, it starts the C++ physics engine over a dynamic link library. As it can be seen on the Figure 3.1, the physics engine accesses the database directly and not over the user interface in visual basic. This provides a faster execution of the engine.

3.4 Prior Results

In prior results using the sportsSemble nodes before having this database system, we were limited to direct interpretations of the measurements taken by the individual nodes of the system [Lap08]. The data for individual nodes was fused together into a vector

representation. These results were done manually, without having a database engine. Therefore, it was a very time consuming procedure, which took several hours. With the database, this can now be achieved within about 30 seconds.

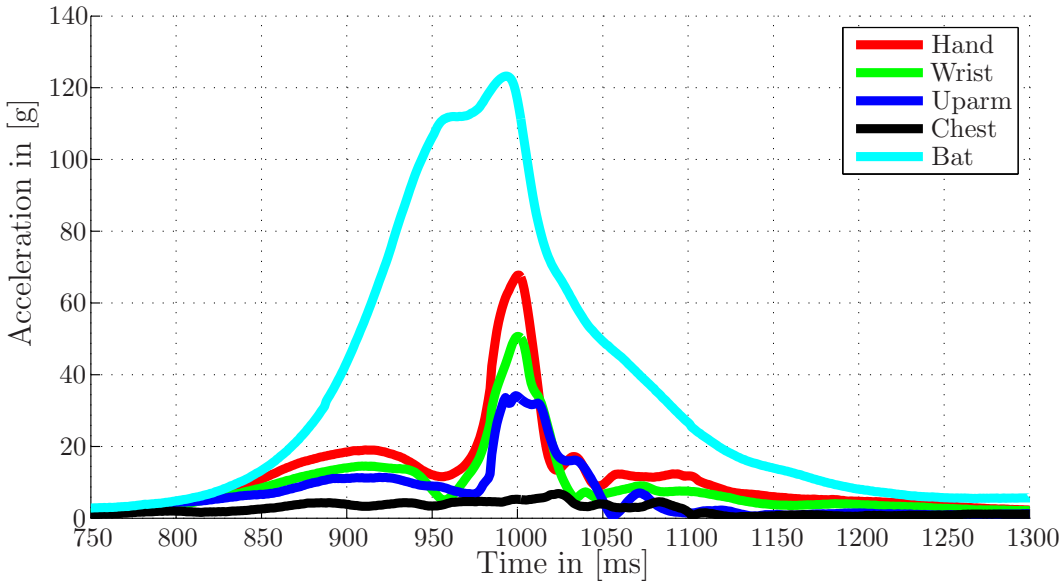

Figure 3.4: Single Swing Acceleration Data

The goal of this work is to properly fuse the data and to agilely evaluate it using a physics body model. This data fusion allows a much more precise data analysis, and therefore also much better data interpretation. It allows data correction by the use of mathematical estimators and therefore reduces sensor noise. For example, Figure 3.4 shows the results of a batting free swing with up to 120G for the maximum of the batting acceleration at the end of the bat.

4. Data Fusion

Although "Data Fusion" has broad interpretation[1], our specific implementation here is to use the accelerometer, gyroscope, and compass data to estimate the movement of a human being. The onboard digital compass is used to find the initial orientation of the player and to correct errors in the position that is estimated using the gyroscope data. In this chapter, we describe how we fuse the data using a physics engine. Additionally, some mathematical models are presented, which reduce the integration error of the inverse kinematics solver in the physics engine.

4.1 Combination with a Physics Engine

A physics engine is a computer program that exploits basic physical laws (primarily mechanics) to predict real-world motion under given conditions. These engines allow the programmer to create a real life or fantasy world for applications ranging from gaming to scientific analysis. Perhaps the currently best-known fantasy world, for example, is the world rendered in the new movie *Avatar* [tCF09]. In this film (as in most films employing advanced computer graphics these days), physics engines have been used to drive animations, which then exhibit very realistic motion behavior. Both real-time physics engines and high precision physics engines are generally available - the first engine is optimized for rough but fast calculations and the second for precise but slow calculations. The goal of both in this application is to improve the realism of the virtual world. For real-time physics implementations, it is necessary that the calculations are done faster than the user can react (e.g., 30 ms or better). In this case, it is not possible to implement high precision physics, because it is very compute-intensive. Therefore this project uses a real time physics engine.

Physics engines have generally two main components, which are the collision detection and the dynamic simulation. The collision detection is necessary to prevent objects from interleaving, and the dynamic simulation is responsible for resolving the forces between objects. Modern physics engines also implement capabilities such as fluid simulations, flexible object simulations, animation control systems and asset integration tools.

Three different paradigms are used when implementing physics engines [Gar06]. The impulse based methods apply the basic law of Newton stating that the sum of the impulses has to be zero. In the constraint based methods, a system of constraint equations are solved that estimate physical laws. And finally, interactions modeled as mass spring systems are called "penalty methods" - this type of method allows deformable or soft body physics. The type of physics engine used in this thesis is capable of simulating all three of the paradigms. In our application, however, only the constraint based method is used, as

[1] Set of methodologies for fusing information coming from different, and sometimes non homogeneous, sources. [Wik10]

the body segments can be modeled as rigid bodies connected to each other over joint constraints.

In order to simulate the behavior of a human body, a skeleton model surrounded with a mesh grid is used. This skeleton model is also called a ragdoll model. It is composed of rigid bodies tied together by mechanical constraints. The joints between the bodies or bones are limited in their range of movement in order to simulate the physical limitations of a real human body.

4.1.1 Engine Choice

For this project, the physics engine had to be adaptable enough to allow for two things. The first was the ability to input real accelerations and angular velocities, as measured by our sensors. The second was to allow the definition of constraints for each joint. Additionally, as the joints of the human body are very complex, the physics engine needs to provide the ability to define complex joints. Initially OpenSim [htt09] was evaluated, however it was not capable of having a stable joint definition. Eventually the Intel physics engine, Havok [cCTRLtH09] was chosen, as it is much more developed. Havok allows creation of complex joint definitions as well as providing for import and export of physical data, such as forces at joints or rates and accelerations at various points.

4.1.2 Usage of the Engine

Our idea is to drive the bones (rigid bodies) of the skeleton model by the measured accelerations and angular velocities that were recorded by the nodes mounted on the body and stored in the database (Figure 4.2). This model, when paired with the physics engine, should ideally reproduce the real-life movement in the virtual world of the engine. This approach also permits calculation of torques and forces that occur at the joints of the bones, and provides a history of joint angles over time.

Normally a real time physics engine updates its current state every 33 ms to service a frame rate of 30 frames per second. In our case, the sensors sample with 1000 Hz, therefore the physics engine is configured to update every millisecond - as our application runs on cached data, real-time performance is not necessary.

Implementation

To get the first initial position, an algorithm has been written to frame a burst of motion by identifying the points around the activity of interest when the athlete is not moving. The algorithm first identifies the peak inertial movement of the athlete across an athletic gesture like a pitch or swing then steps backwards in time from this point to find the closest point in time where the athlete is not moving. Stepping backwards from the peak is necessary, as the player is probably moving at the start of the time frame where the sensor system is recording. If the algorithm were to start from the beginning of the recording frame, the initial time would be probably far away from the movement of the athlete. In this case, the

errors accumulated from the sensors during the time until the athlete moves will probably distort the initial position and therefore add significant error to the simulation.

The algorithm then calculates the root mean square (RMS) of the accelerometers and gyroscopes. These RMS values than get compared to $9.81\frac{\mathrm{m}}{\mathrm{s}^2}$ for the accelerometers and to zero for the gyroscope. Because even if the accelerometer is not aligned with the vertical the RMS of the accelerometer data has the value of the length of the vector, which is 1 g on the earth. If this length of the gravity vector is different, we have to assume that the athlete is in motion. The algorithm scans the data for regions where the RMS is within a tolerance of one percent (a higher threshold is used if no time frame can be found where there is no movement).

The same algorithm is used to find the end of the movement. This allows the analysis to be run only on the data that is of interest.

4.1.3 Inverse Kinematics

Most of the physics engines use the Jacobian transpose method [Bus04] to reduce the computational costs of the inverse kinematics computation. To properly explain this method, first the forward kinematics and the Jacobian matrix need to be explained and then the inverse kinematics problem can be addressed.

Forward Kinematics and Jacobian Matrix

A multi body system of rigid bodies is modeled with a set of segments connected by joints. There are several different joint types, all having one thing in common; to limit one or more of the 6 degrees of freedom (6DOF) of the two bodies in relation to each other. A hinge joint, for example, limits two bodies to a single axis rotation to and from each other. Along this axis, the bodies can not exhibit mutual force or torque along the axis of the joint, as their movement on this axis is allowed by the joint.

In order to understand the inverse kinematics, it is important to get to know some of the scientific terms. Each rotational joint is described by a *joint angle*, θ_i, and the *end effector position*, s_i, which is a function of the joint angles and the *target position* t_i. The end effector position is the position and orientation of the last joint in a chain of joints. The multi body (numerous rigid bodies related to each other) will be controlled by specifying target positions for the end effectors. The desired change in position, e_i, is therefore $t_i - s_i$.

When we consider only rotational joint angles we can express $\vec{s} = \vec{s}(\theta)$. So our inverse kinematics (IK) problem is to find the angle θ so that

$$t_i = s_i(\theta) \tag{4.1}$$

Although there might not be a unique solution of this problem. It is possible to select to a good solution. In order to do this, the function s_i is approximated by the Jacobian matrix. The Jacobian matrix is a function of the joint angles θ

$$J(\theta) = \left(\frac{\partial s_i}{\partial \theta_j}\right)_{i,j} \tag{4.2}$$

$J(\theta)$	Jacobian matrix
e_i	Change of position
t_i	Target position
s_i	End effector position
θ	Joint angle

Hence we obtain a vector with $k \times n$ entries of $\mathbb{R}^3$ vectors (three dimensional real vectors), where k is the number of end effectors s_i an n the number of joints.

The forward equation of this system can be written as

$$\dot{\vec{s}} = J(\theta)\dot{\theta} \tag{4.3}$$

The Jacobian presents an iterative linear "tangent" method for solving Equation 4.1. Since at each time step we have θ, $\vec{s}$ and $\vec{t}$, the Jacobian matrix can be computed (Equation 4.2). By incrementing or decrementing the joint angle θ we are now able to find the best solution of the problem (Equation 4.1). The change of the end effector position can be approximated using the Jacobian matrix and the change of the joint angle

$$\Delta\vec{s} = J(\theta)\Delta\theta \tag{4.4}$$

$\Delta\vec{s}$	Vector of end effectors
$J(\theta)$	Jacobian matrix
$\Delta\theta$	Change of the joint angle

The change of the joint angle is basically chosen in order to get $\Delta\vec{s}$ equal to $\vec{e}$. Another approach is to chose $\Delta\theta$ to match the velocity of $\Delta\vec{s}$ of the target position.

The update of the joint angle can be chosen in two different modes:

- At each simulation step the joint angle performs a single update, so that the end position follows the target position.
- The joint angles are updated iteratively until the end position is sufficiently close to the solution.

Hybrid systems are also feasible using both of these approaches.

Compute the Jacobi Matrix

We consider the jth joint a hinge joint with a single degree of freedom with a joint angle, θ_j. Let us call p_j the position of the joint and v_j be a unit vector along the current axis of rotation. In this case if the ith end effector is affected by the joint, then the corresponding Jacobian entry is [OS84]

$$\frac{s_i}{\theta_j} = v_j \times (s_i - p_j) \tag{4.5}$$

Here we can see, that if the jth joint does not affect the ith end effector, the corresponding entry of the Jacobian matrix is zero.

If we consider the jth joint to be only translational, the entry of the Jacobian matrix is even more obvious to compute. If we consider v_j the vector along the translation of the

joint, the joint "angle" measures the distance moved in direction v_j. In the case if the ith end effector is affected by the jth joint, we get

$$\frac{s_i}{\theta_j} = v_j \tag{4.6}$$

p_j	The jth position of the joint
v_j	The jth unit vector along the current axis of rotation
θ_j	The jth joint angle
s_i	The ith end effector position

Jacobian transpose method

The problem now is that there can be situations, where the kinematic problem can not be solved and building the inverse of the Jacobian matrix takes a lot of computational power. There are several methods of addressing this problem. Each of these methods has its conveniences and its drawbacks. In this context only, the Jacobian transpose method is presented, as it is the most common solution.

The basic idea of the Jacobian transpose method (Equation 4.7) is very simple. It just uses the transpose of J instead of the inverse of J. This equation is true for some appropriate scalar α.

$$\Delta\theta = \alpha J(\theta)^T \vec{e} \tag{4.7}$$

$J(\theta)^T$	Transposed Jacobian matrix
$\vec{e}$	Vector of the changes of position
$\Delta\theta$	Change of the joint angle
α	Scalar

It is true that the transpose of the Jacobian is not the same as the inverse, but a mathematical theorem allows us that estimation.
Theorem 1: For all $J(\theta)$ and $\vec{e}$, $\left\langle J(\theta)J(\theta)^T\vec{e}, \vec{e}\right\rangle \geq 0$.
Proof: $\left\langle J(\theta)J(\theta)^T\vec{e}, \vec{e}\right\rangle = \left\langle J(\theta)^T\vec{e}, J(\theta)^T\vec{e}\right\rangle = \left\|J(\theta)^T\vec{e}\right\|^2 \geq 0$.

That means, that if we change a small value of $\alpha > 0$, we will change the end effector position by $\alpha J(\theta)J(\theta)^T\vec{e}$ (Equation 4.3). Now we just have to minimize alpha by setting the first derivation to zero and we have made a good approximation of our inverse kinematic problem. Of course this is just an approximation of the problem, but it is adequate for our application and much more effective than building the inverse.

4.1.4 The Skeletal Model

The skeletal model or ragdoll model in a physics engine is the most interesting part for this project. As we want to simulate the behavior of the arm of a pitcher during a throw, it is necessary to model the arm as accurately as possible (Figure 4.2). The constraints of the physical arm model limit the movement, but should not restrict anatomically possible movements.

Although not strictly anatomically accurate (e.g., most joints, especially the shoulder, allow for a limited amount of translational as well as rotational freedom as the extend), our model of an arm, adopts basic joints like the hinge or ball joint. The human arm allows some movements that are not allowed by a hinge joint, but the ball joint does not restrict any rotation. Therefore, in addition to the joint model, it is necessary to define areas where the object cannot be moved. For the shoulder for example it is possible to do all movements within defined cones (Figure 4.1). These cones can be created in all three dimensions in order to represent a good estimation of the shoulders range of motion. Granted, in reality these stops are soft and "springy", but for the purposes of this initial limited analysis, we assume hard stops here.

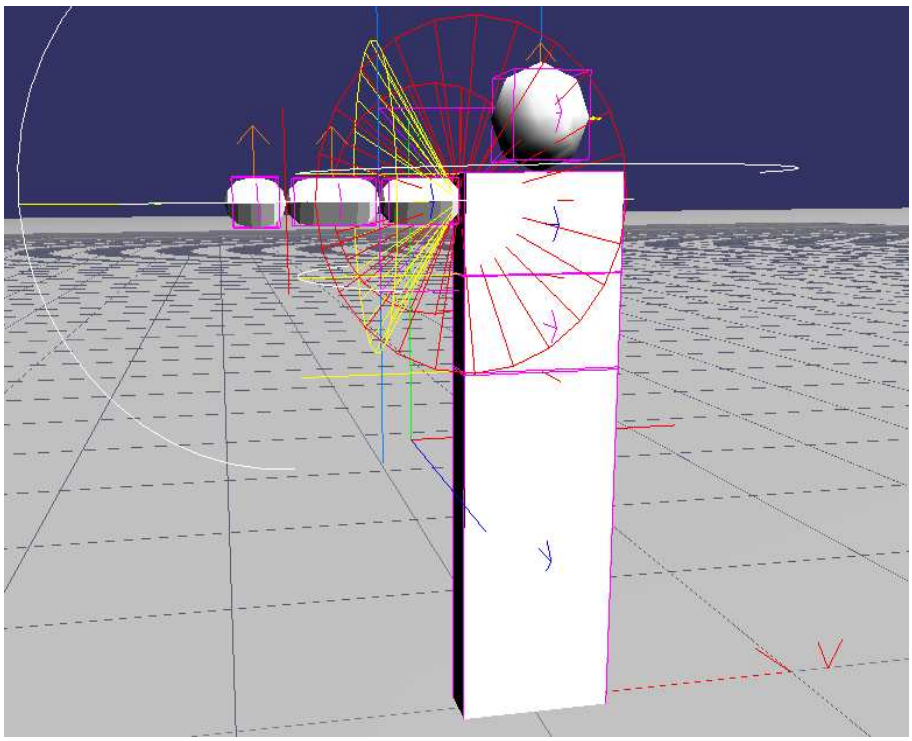

Figure 4.1: Shoulder Model

The physics engine uses these constraints together with the given forces and torques to move the object in a correct anatomical way.

The idea is to use the biomechanical knowledge of the arm constraints and limit the movement of the arm by these constraints. However, it is necessary to define the constraints that are not too strict, because important information of the movement can be lost.

For medical applications, understanding these extremes is the first step in understanding shoulder injury. Therefore it is important to still be able to plot out the forces and torques in the constrained direction. As the sensor system provides accelerations and angular velocities directly, we are able to calculate out the forces in between of the joints before the constraints are applied by the physics engine. The constraints then just help to avoid

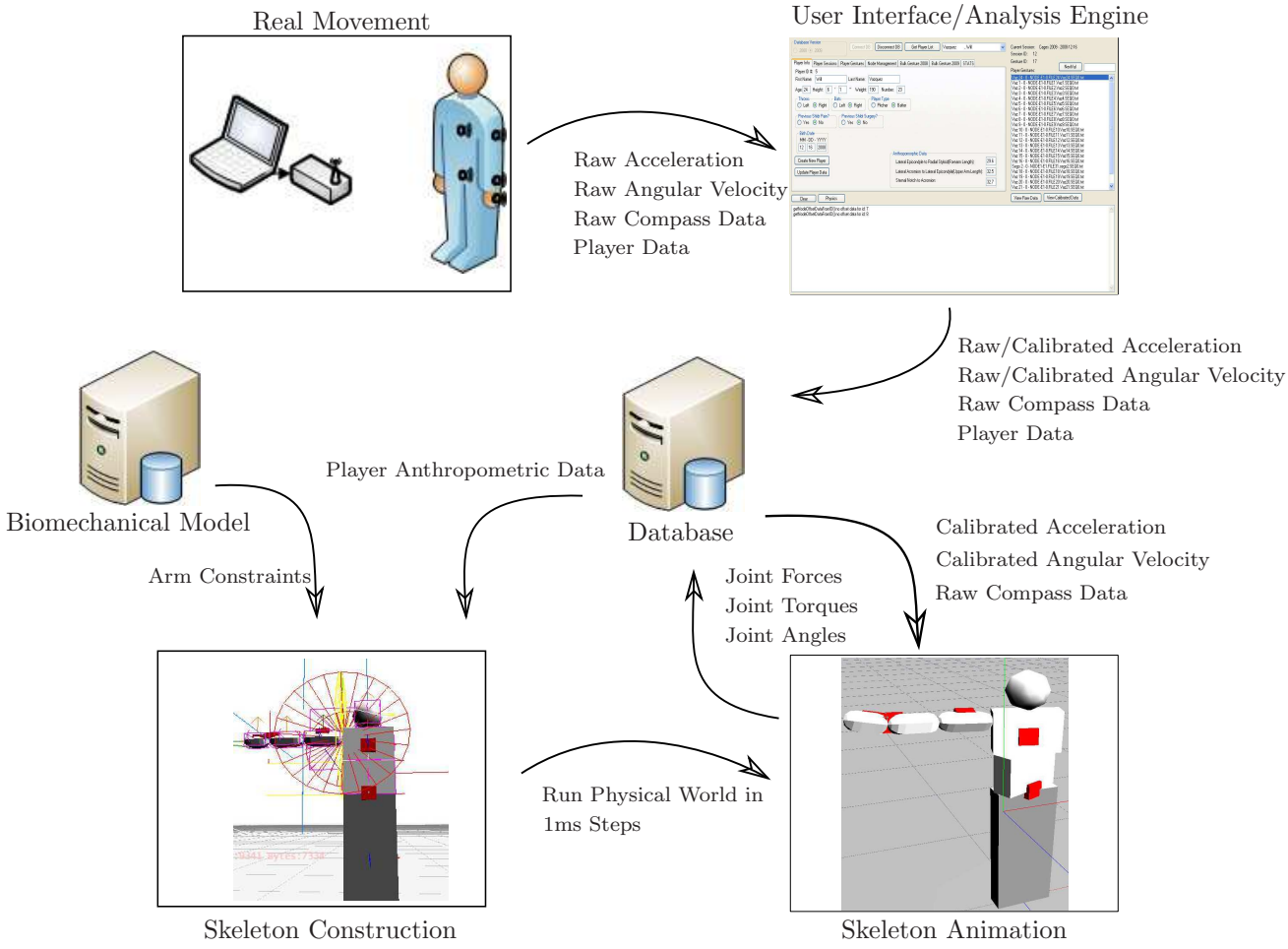

Figure 4.2: Skeleton Construction

unrealistic movements. The physics engine just provides the mathematical simulation framework that accommodates the mass, orientation, angular and linear velocity of each segment. Section 4.1.6 explains how the forces and torques are calculated in the physical simulation.

Waist Model

The waist is only very roughly approximated in this model, as the main goal is to measure the forces in the arm joints. Therefore, only a rotation about the center of the body is possible.

Shoulder Model

As explained above, the shoulder is modeled as a restricted ball joint. The restrictions can be defined in form of restriction angles about one rotation axis. All three rotation

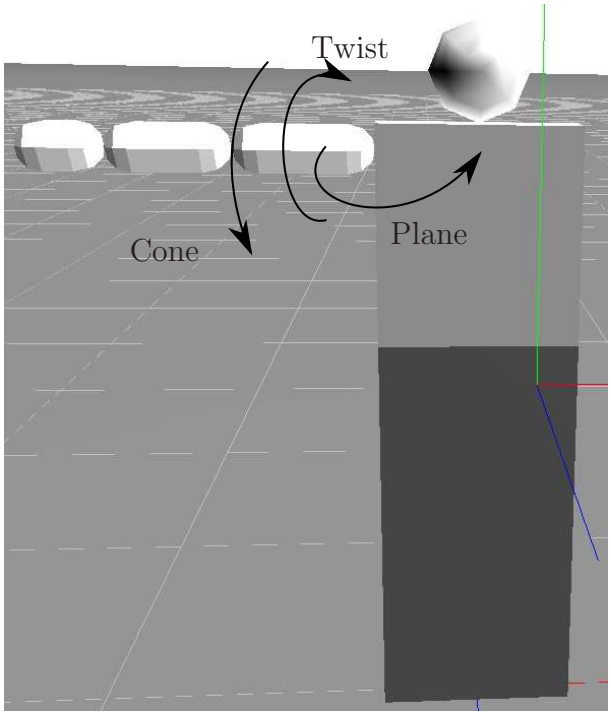

Figure 4.3: Shoulder Model

axes can be restricted in this way. In this way the model cannot move in a manner the human shoulder is not able to. Although, the International Society of Biomechanics (ISB) recommendation on definition of coordinate systems of various joints for the reporting of human joint motion [GW05] defines a much more complicated shoulder model, the shoulder model here defines the shoulder motions in an easier but also precise enough way, at least for the initial study reported in this thesis.

For the limitations on shoulder rotation the cone, twist and plane axes of the shoulder are limited by the following values.

- Cone Angular limit $\left[\frac{-\pi}{2} - \frac{\pi}{16}\right]$
- Twist Angular limit $\left[\frac{-\pi}{2}, \frac{\pi}{2}\right]$
- Plane Angular limit $\left[-\frac{\pi}{4}, \frac{7\pi}{8}\right]$

Figure 4.3 shows the three rotations of the shoulder. The initial reference of the shoulder is chosen as in the figure. That means, that all angles are defined relative to this position.

Elbow Model

At first view, the elbow seems to be a limited hinge joint with a maximum range of 0 to π. However, the elbow joint is actually much more complicated as there can be a 90 degree rotation between the Radius and Ulna bones. Therefore, it is not sufficient to model the elbow as a limited hinge joint. In the model adopted for the elbow, a limited ball joint has also been chosen, as it allows also a twist rotation between Radius and Ulna.

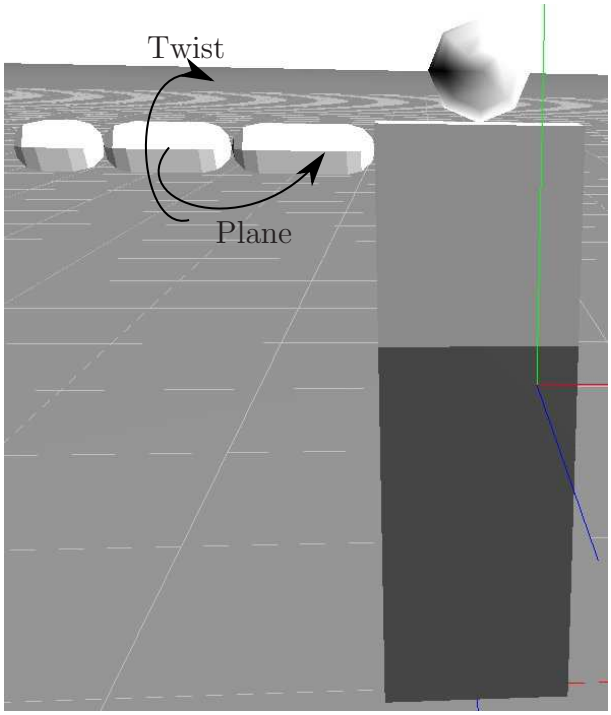

Figure 4.4: Forearm Model

- Twist Angular limit $\left[\frac{-\pi}{2}, \frac{\pi}{2}\right]$
- Plane Angular limit $[0, \pi]$

Figure 4.4 shows the plane and twist axes of the elbow joint.

Wrist Model

In order to keep the model as simple as possible, the wrist joint is modeled as a limited hinge joint. Of course this is a rough approximation, as there can also be rotations to the side. However, it was necessary to keep the model simple and therefore the wrist joint is modeled as a limited hinge. The joint angle limitation is from minus to plus $\frac{\pi}{2}$ around the axis shown in Figure 4.5.

4.1.5 Application of the Accelerations and Angular Velocities

The standard approach of camera based motion tracking systems is to apply positions and orientations to a skeleton. This allows a pretty animation of the body movements and is the perfect method for animations. In our case, the sensors do not give positions and orientations, but only information on how the orientation and position is changing. However, together with the digital compass, which gives a global orientation, the orientations and positions can be derived by integrating the inertial sensors over the time, correcting with compass values.

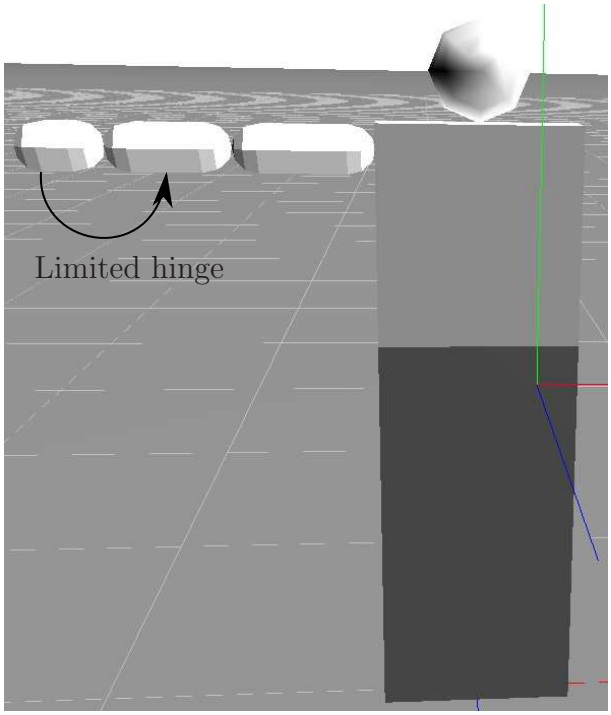

Figure 4.5: Wrist Model

The whole idea behind this thesis is to apply forces and angular velocities at the body segments. This allows us to get precise information about what's happening in the joint segments. In several current applications the Kalman filter algorithm has been used to fuse magnetometer, accelerometer and gyroscope data together to get the orientation of the body segments [LEJP09]. This again allows for the making of nice animations, but gives no kinematic information as the segments just get simultaneously reoriented and repositioned to fulfill the joint constraints.

The physics engine allows us to apply real forces, sensed directly by the nodes, and change the angular velocities of the segments. Both functions (`applyForce`, `setAngularVelocity`) change the velocity stored in the segment class. A force changes the linear velocity of the segment to which it is applied, but also affects the angular velocity if it is not applied at the center of gravity. As we are not able to measure at the center of gravity, it is necessary to know the position of the measurement to be able to apply the force exactly at this point. The physics engine than allows us to apply the force exactly at this point and creates the changes in the linear and angular velocity dependent on the point, where it is applied. To get the force, the author used human body statistics [Win05], which give information on the mass of each segment depending on the total body weight. The percentage of the body mass then is applied to each segment to define the skeleton of the body. Now we have all information necessary to apply the force on a segment (Code 4.1.5). The subject's weight is known and multiplied with the segment mass to give the force. This, combined with the relative sensor position on the segment, allows us to apply the force.

```
1  mass = Ragdoll->GetSegment(NrSeg)->GetRigidBody()->getMass(); //get the mass of the
       segment which is to move
   q = Ragdoll->GetSegment(NrSeg)->GetRigidBody()->getRotation(); //gets the segment
       rotation
3  q.mul(Ge->Nodes[NrSeg]->q_off); //applies the node offset on the orientation
   rot_acc.setRotatedDir(q,*acc); //rotates the acceleration in world space
5  force = rot_acc;
   force.mul4(mass); //multiplies the rotated acceleration with the segment mass
7  Ragdoll->GetSegment(NrSeg)->GetRigidBody()->applyForce(timestep , force, Pos); //applies
       force
```

To apply the angular velocity, less information is necessary, as it is not dependent on the segment mass and also not dependent on the relative position on the segment. Therefore it is just necessary to update the angular velocity of the segment 4.1.5.
For both implementations, it is necessary to apply the acceleration and angular velocity vector in body space as they are measured in the body space.

```
1  q = Ragdoll->GetSegment(NrSeg)->GetRigidBody()->getRotation(); //get the mass of the
       segment which is to move
   q.mul(Ge->Nodes[NrSeg]->q_off); //applies the node offset on the orientation
3  rot_w.setRotatedDir(q,*w); //rotates the angular velocity in world space
   Ragdoll->GetSegment(NrSeg)->GetRigidBody()->setAngularVelocity(rot_w); //sets angular
       velocity
```

These two functions described above give a model in motion. The two functions allow us to fuse the linear and angular velocity to get positions and orientations.

4.1.6 Calculation of the Forces and Torques

The application of a force of a segment in the physics engine changes the velocity of a the segment instantly. That means the force applied to a body multiplied by the time it is applied on the body gets added to the actual linear velocity of the body. The same is valid for the angular velocities of a segment. In our case, we measure the angular velocity directly and set it on the segments.

It is possible to measure the change of the velocity before the solver of the system is applied. The solver checks the constraints of the system and changes the velocities in order to not violate them. The inverse kinematic solver creates reactionary forces to fullfil the joint constraints. Therefore, it brings an error in our measure. In order to prevent this error, the force gets computed just before the solver step. This allows us to utilize the models constraints and at the same time compute the correct forces acting on a joint.

Figure 4.6 should explain graphically the idea mentioned above. This figure shows the time line of the physical simulation. At the first step, the linear acceleration and angular velocity get applied. This immediately changes the old linear, $v[t-1]$, and angular velocities, $\omega[t-1]$, of the segment. It is important to note that there is no change in the velocities based on the constraints as that solver step comes later in time. Now the difference of the velocity is used to compute the force and torque of the segment (Equation 4.8). The force, F, is the change of the velocity, Δv, divided by the change of the time, Δt. The change of the time is in our case equal to the time step of the engine.

$$F = m \cdot \frac{\Delta v}{\Delta t} \tag{4.8}$$

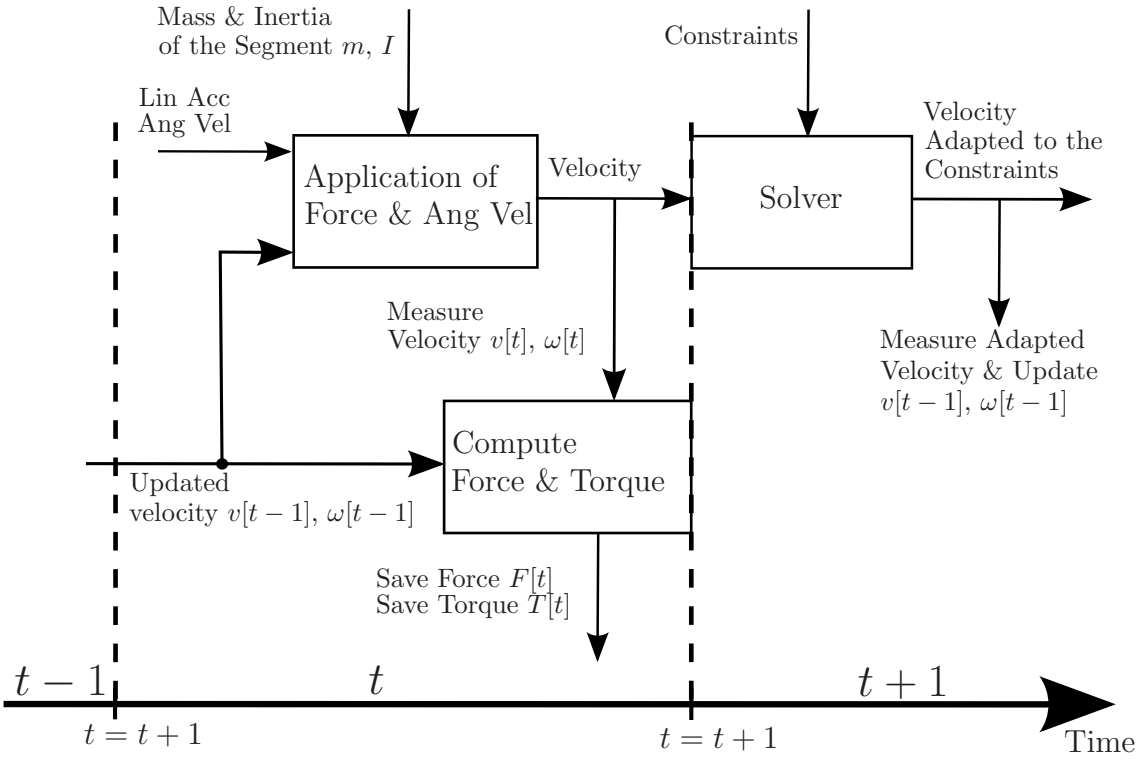

Figure 4.6: Timing of the Force and Torque Calculation

F	Force
m	Mass of the Segment
Δv	Change of the linear velocity
Δt	Change of the time

After the solver step, this velocity gets updated by the velocity value that is adapted to the constraints. This new velocity fulfills the constraints of the system. In this way the force for the next step can be calculated using the updated velocity after the last solver step and the new velocity before the next solver step, which takes into account the new accelerations and angular velocities.

The same procedure is used to compute the torque. The only difference is that the gyroscope directly measures the angular velocity and not the angular acceleration. This makes the calculation even easier, as we can directly change the angular velocity by overwriting the old angular velocity. The torque (Equation 4.9) is defined as the multiplication of the matrix of inertia, J, times the first derivative of the angular velocity, $\dot{\omega}$, which can be also expressed as the change of velocity over time.

$$T = J \cdot \dot{\omega} = J \cdot \frac{\Delta\omega}{\Delta t} \tag{4.9}$$

T	Torque
J	Matrix of Inertia from the center of the joint

$\dot{\omega}$	First derivative of the angular velocity
$\Delta\omega$	Change of the angular velocity
Δt	Change of the time

In order to construct a system of multiple joints, this formula gets more complicated, as the matrix of inertia has to be translated in space. In this case the physics engine is again really helpful, as there is an opportunity to translate the inertia matrix by using the distance between the center of gravity of the segment and the center of the joint. All that is necessary to add the mass of the segment times the distance to the joint to the accurate positions of the matrix of inertia.

$$J_{ij} = I_{ij} + m(a_l a_l \delta_{ij} - a_i a_j) \tag{4.10}$$

J_{ij}	Entries of the matrix of inertia around the joint
I_{ij}	Entries of the matrix of inertia around the center of gravity
m	Mass of the segment
$\vec{a}$	Displacement vector $(a_1 x + a_2 y + a_3 z)$ from the center of gravity to the new axis
δ_{ij}	Kronecker delta

Equation 4.10 shows the translation of the inertia by the vector a. This so called parallel-axes theorem [Bög06] allows us to translate the matrix of inertia the torque in space. This means that the torque can be calculated at any joint by using Equation 4.10 and Equation 4.9 for the calculation of the torque. This is an easy way to compute the torque, as the angular velocity around the center of gravity of each segment in world coordinates is stored within the segment class. This finally enables a fast algorithm that integrates the torques for each joint.

4.1.7 Software Structure of the Physics Model

To understand the structure of the physics model, a UML class diagram will be analyzed here. This diagram (4.7) shows the relations between the different classes of the physics program.

CallWorld and Physics_World

The class, that constructs all the other classes is the CallWorld class, which constructs the Physics_World class, which creates a physical environment with an earth acceleration. In this class, the constructor creates all the other objects of this program. It has one main function, which is called *RunWorld()*. During the execution of this function, the physical behavior of objects in the physical space will be simulated. During this simulation this function calls the *movehigh()* and *moveanghigh()* function, which drive the movement of the physical objects related to the sensor data. At the end of each simulation step (every millisecond), the function calls a function for each segment of the skeleton to store the calculated impulses and forces at each segment. Every 150 milliseconds, the Kalman filter gets applied by the function *Apply_Kalman()*.

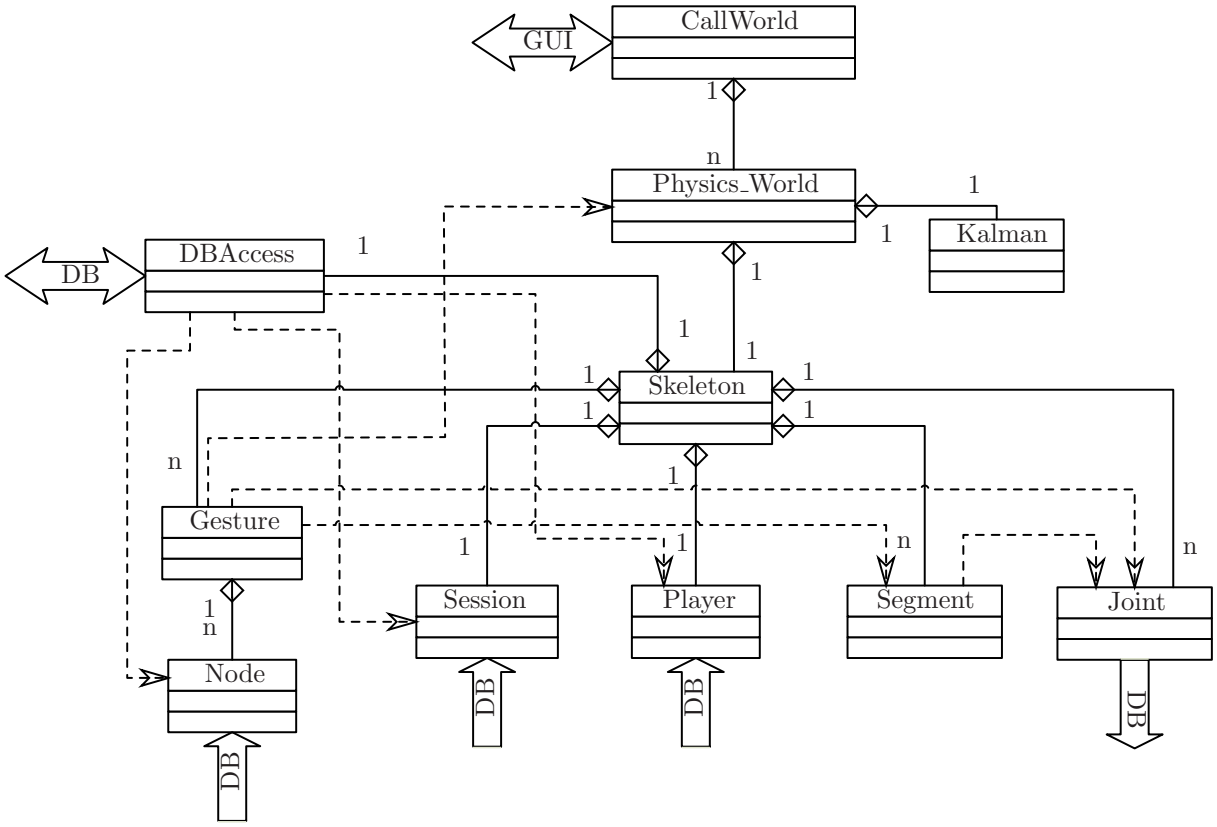

Figure 4.7: Physics Model Class Diagram

In this environment, a skeleton is constructed, that is represented by the skeleton class. This skeleton uses some other classes to get the information on where to place the physical objects, such as arm chest and leg, as well as how to move them during the simulation. The skeleton class creates all other classes, which are related to the size, structure and movement of the model. At first, it creates the session and the player, which give information about the player's anthropometric data and the node placement. Out of this information the segments of the skeleton can be created. Then it builds the classes gesture and node, which give information on which sensors are used and what data the sensors gave. One gesture is at the moment measured by five nodes, but this can change depending on the size of the sensor array that we use in our tests. Finally, the class skeleton creates the class segment and joint to save all the joint related information such as the joint angles, impulses, forces and torques. This information will be saved in the database when the simulation is over.

Some functions get applied in the physics world class to first construct an object *Ragdoll* of the skeleton class, then to fix the initialization time (*Get_Inittime()*) and the end of simulation time (*Get_Endtime()*). How the begin time and the end of simulation time is computed will be explained in the skeleton class section 4.1.7.

In the *RunWorld()* function, *begintime* sets the start of the for loop and *endtime* the end

of it. During the simulation, at every time step the functions *Update_Rel_Sensorpos()* and *Set_JointPosAngle(...)* gets called. These two functions update the relative sensor position on the arm segments and the joint position and angle. After applying the movement functions, the function *CalcImpForce()* of the segment and the function *Calc_Torques()* get applied to compute all the information on torques and forces before the solver gets applied at the next time step. After this solver time step, the linear and angular velocities are updated by the function *Reset_Vels(...)* as explained in Section 4.1.6. The other functions applied in the *RunWorld()* function are only for visualization and the time synchronization.

Skeleton

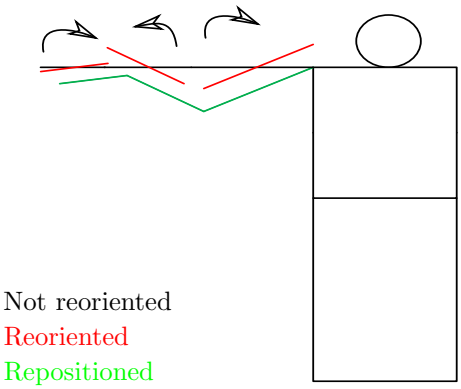

Figure 4.8: Problem of repositioning the segments

The class Skeleton is the main class of the program. It calls all the constructors of the other classes and builds the middle of the UML class diagram 4.7. It constructs:

- The class DBAccess
- The class Player
- The class Session
- The class Gesture
- The classes for the Segment
- The classes for the Joint

It also constructs the segments of the skeleton and their constraints. It reduces the drift error by computing the initialization point as close as possible to the maximum of the gesture. In order to find this point the algorithm first detects the maximum of the angular velocity, then steps backwards from it to find a point where the athlete was not moving. This point gets computed by comparing the root mean square (RMS) values of the acceleration and the angular velocity. For the acceleration, the RMS value gets compared to the gravity and the angular velocity gets compared to zero. As the sensors are not perfect, and even after a good calibration have some offset, it is necessary to raise the limits every time no value is found. Accordingly, the algorithm repeats this procedure until the threshold reaches 50% and then issues an error message. This algorithm is used to find the begintime (*Set_Inittime(...)*) and the endtime (*Set_Endtime(...)*) of the gesture. As we know that at the inittime the athlete was not moving or barely moving, the compass data can be used to set the initial position with a call of the function *Set_Initangle(...)*.

As the setting of the initial angle of the segments can violate the segment constraints, the segments have to be retranslated (Figure 4.8). The figure shows a 2D example of this problem, but it should be easy to imagine this problem projected into 3D.

DBAccess

The class DBAccess can be seen as a gateway to the database. Every function of this class connects to one part of the database over a SQL query and either selects some data from the database or inserts some data into the database. In order to keep it simple, all other classes that need to access the database have to use the class DBAccess.

Player and Session

Player and Session are two classes that represent the same structure than in the database. They are only used to store the data in the same way as within the database.

The class Player saves the data of the player and his anthropological data. The class Session saves all the data of the player session as well as some statistical data from this session.

Gesture and Node

The class Gesture can be seen as the mother class of the nodes. It creates and deletes all the classes Node. In the class node, all information about the movement is stored including the linear accelerations and angular velocities in low range as well as in high range and the compass data. This class can be seen as the class that keeps the data of the movement. In order not to have the need to rotate the coordinate system every time step of the simulation, the low and high range data gets directly inserted in the right orientation. This means that the nodes for the arm are inserted differently than the nodes of the upper body, as they are oriented differently.

The following contribution is used for the arm:

- $X_{PhWorld} = Z_{Reality}$
- $Y_{PhWorld} = Y_{Reality}$
- $Z_{PhWorld} = -X_{Reality}$

The following contribution is used for the upper body:

- $X_{PhWorld} = -X_{Reality}$
- $Y_{PhWorld} = Z_{Reality}$
- $Z_{PhWorld} = Y_{Reality}$

For the compass the data gets inserted as heading, pitch and roll without considering the node's orientation, as this is currently not possible unless a quaternion gets directly created. The problem with the compass data is that the pitch and roll depend on the direction of the heading. This means with a change of the heading, the pitch and the roll have to be rotated equivalently.

Segment and Joint

Segment and Joint are the two classes that store the data of the impulses, forces, torques and joint angles. The class Segment stores the forces, impulses and angular velocities of each segment and the class Joint uses this information together with some other segment data to compute the force and torque as described in Section 4.1.6.

4.2 Kalman Filter

The Kalman filter provides the ability to reduce the effects of drift in the accelerometer and the gyroscope data by using optimal estimation to calculate orientation. To understand the general idea of this filter algorithm it is best to start explaining it using its final Equation 4.11.

$$\hat{X}_k = K_k Z_k + (1 - K_k)\hat{X}_{k-1} \tag{4.11}$$

$\hat{X}_k$	Current Estimation
$\hat{X}_{k-1}$	Previous estimation
K_k	Kalman gain (averaging factor)
Z_k	Measured Value (observation)

The Kalman filter finds the most optimum averaging factor for each consequent state. The averaging factor is a weight assigned to the next set of data. The filter remembers the last state, and combines it with the actual measured data. What the algorithm describes is how to compute this Kalman gain in order to get the best-estimated value.

4.2.1 Kalman Equations

The Kalman filter is not a simple filter; it is probably easier to understand when it is called an estimator. The Kalman algorithm is used derive an optimal filter that estimates parameters driven by a process with multiple inputs perturbed by white noise. These inputs are fused together to obtain the estimation. The Kalman filter finds the most least-square-optimum averaging factor for each consequent state. The best way to understand the Kalman filter is to understand the problem we want to solve. The optimization problem can be defined as calculating a data point such that the real value, x_k, minus the estimated, $\hat{x}_k^+$, ideally gives zero [MS01] - more precisely, that the least-square sum of the residuals between real and estimated values is minimized.

$$E\left\langle \left[x_k - \hat{x}_k^+\right] z_i^T \right\rangle = 0,\ i = 1, 2, ...k \tag{4.12}$$

The problem is that we don't know the real value. All we can do to get the real value is to measure, but every measurement is to some degree defective and has some noise effecting it. Therefore it is necessary to use an estimator to find the real value or at least to approximate it.

The Kalman filter uses past states to estimate the actual state. It consists of a system of states, x_k, and observations, z_k. The two main equations of the Kalman filter are the following (Equation 4.13, 4.14).

$$x_k = Ax_{k-1} + Bu_k + w_{k-1} \tag{4.13}$$

$$z_k = Hx_k + v_k \tag{4.14}$$

x_k, x_{k-1}	Actual and previous state
z_k	Measrued Value (observation)
u_k	Control inputs
w_{k-1}	Process noise
v_k	Measurement noise
A	State time evolution
B	Control to state space conversion
H	Input to state space conversion(measurement sensibility matrix)

The first equation describes the system with its states and the second describes its measured values. Our process, just like our measurement, is noisy, and this noise is expressed by w_{k-1} and v_k. In our approximation we make the assumption that these two noise soucres are Gaussian and independent from each other. Therefore, the expected value of their multiplication is equal to zero (Equation 4.15).

$$E\left\langle w_k v_j^T \right\rangle = 0 \quad \text{for all k and j} \tag{4.15}$$

Furthermore, we assume that we don't have an offset error, so the two Gaussian's are symmetric to the origin. Their probability distributions can be expressed with the following expressions (Equation 4.16, 4.17).

$$p(w) \sim N(0, Q) = \frac{1}{Q\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x}{Q}\right)^2\right) \tag{4.16}$$

$$p(v) \sim N(0, R) = \frac{1}{R\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x}{R}\right)^2\right) \tag{4.17}$$

In practice, the process noise covariance, Q, and measurement noise covariance, R, matrices might change with each time step, however here we assume they are constant. The process noise covariance is equal to the variance of a Gaussian distribution. This assumption allows us to derive the process and measurement noise covariance [XYERB06]. As the estimated value, μ_x, is zero, the autocovariance, Σ_{xx}, is equal to the autocorrelation, Φ_{xx}. Therefore, we are capable of estimating the autocovariance by the variance of the noise times the identity matrix, I (Equation 4.18).

$$\Sigma_{xx} = \Phi_{xx} - \mu_x\mu_x = \Phi_{xx} = \sigma^2 I \tag{4.18}$$

Using these approximations we can directly compute the Kalman filter. The filter algorithm uses two steps to compute the approximation.

The first step is the time update, which computes the a priori value, $\hat{x}_k^-$, of the next state and calculates the a priori error covariance P_k^- (Equation 4.19). In this context, the a priori predictions are declared with a minus sign in the exponential and the a posteriori predictions with a plus sign in the exponential. The a priori values are predicted values, which still need to be corrected.

The next step is to incorporate the observations, also called the Kalman step. This step corrects the predicted values by incorporating the measured values. First, the Kalman matrix, K_k, gets computed and the error covariance, P_k, gets calculated. In this step the measured results get incorporated in the approximation of, $\hat{x}_k$, (equation 4.20). The Kalman matrix, K_k, is used to weight the difference between the observations, z_k, and the a priori predictions, $\hat{x}^-$, and the a priori prediction itself.

$$\text{Time update} \begin{cases} \hat{x}_k^- = A_{k-1}\hat{x}_{k-1}^+ + w_{k-1} \\ P_k^- = A_{k-1}P_{k-1}A_{k-1}^T + Q_{k-1} \end{cases} \tag{4.19}$$

$$\text{Kalman step} \begin{cases} K_k = P_k^- H^T (H P_k^- H^T + R_k)^{-1} \\ \hat{x}_k^+ = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \\ P_k = (I - K_k H)P_k^- \end{cases} \tag{4.20}$$

$\hat{x}$ Estimation of x

$\hat{x}^-$ or $\hat{x}^+$	A priori or a posteriori estimation of x
w_{k-1}	Process noise
Q_{k-1}	Process noise covariance
R_k	Measurement noise covariance
K_k	Kalman matrix
P_k^-	A priori error covariance
P_k	Error covariance

The only unknowns are the process and measurement covariance, but they can be computed using the assumption of the covariance and adapting the variance specifically. The assumption is that the process and measurement noise are independent of each other, white, and with normal probability distributions (Equation 4.15 to autocovariance) [WB06].

The variance can be computed from statistical analysis of the measurements of a constant value. A Gaussian distribution can be computed over these values, which gives us sigma. The square of sigma then provides the variance of the observation.

4.2.2 Kalman Example

A good example through which to explain the Kalman filter is a voltage reading of a perfect constant voltage source. The reading is represented here by the state variable, x_k, with a voltmeter, which is clearly the observation, z_k, having a standard deviation measurement noise, v_k.

In this case, the Kalman system is simplified to a one dimensional problem (better known as the Wiener Filter). As the measured value, z_k, is at the same time the state value, x_k, the matrix, H, is the identity matrix, because H is defined as the measurement sensibility matrix. It even gets easier here, because as the voltage is considered as constant. In this case, there is no state time evolution and therefore the matrix, A, is an identity matrix as well. Therefore, we get a really simple system (Equation 4.21, 4.22). All complicated matrix mulitplications shrink to a simple vector, because the identity matrices can be eliminated as they do not change anything.

$$x_k = x_{k-1} + w_{k-1} \tag{4.21}$$

$$z_k = x_k + v_k \tag{4.22}$$

If we than go through the time update and the measurement update of the Kalman filter, we also get a simple system. The unknowns of the system are only the variables, which get computed during the Kalman operation. The a priori error covariance, P_k, for example can be set to zero for the initialization. Therefore, the measurement noise covariance, R_k, is the only undefined variable of the system as the Kalman gain, K_k, is a function of R_k

and P_k^-.

$$\text{Time update } \begin{cases} \hat{x}_k^- = \hat{x}_{k-1}^+ \\ P_k^- = P_{k-1} \end{cases} \tag{4.23}$$

$$\text{Kalman step } \begin{cases} K_k = P_k^-(P_k^- + R_k)^{-1} \\ \hat{x}_k^+ = \hat{x}_k^- + K_k(z_k - \hat{x}_k^-) \\ P_k = (I - K_k)P_k^- \end{cases} \tag{4.24}$$

All we have to choose are the start values P_0 and x_0. Here, a good start value for the value of x_0 is close to the voltage we measure, but it can also be set to zero. However, the value of the error covariance should not be set to zero, as this would mean that the system has no noise, which is never the case and would make the Kalman filter unnecessary. Therefore, the value of P_0 should not be zero as a start value.

4.3 Quaternion

A quaternion is a four dimensional vector with one real and three complex elements. It is used to describe rotations in a three dimensional space in a mathematically elegant way [Lam], [JCKC92]. The quaternion is relevant here because in the following subsection, a quaternion-based Kalman filter will be developed.

Its vector elements are a real value and three imaginary elements i, j, k. To understand this development it is necessary to know the matrix form of the multiplication of two quaternions (Equation 4.25), which is not the same as a multiplication of two vectors. For the quaternion multiplication, the following symbol $\otimes$ will be used. Quaternion multiplication is not commutative, hence the matrix notation of the vector multiplication is the following.

$$p \otimes q = -p \cdot q + p \times q = \begin{bmatrix} q_z & q_y & -q_x & q_\omega \\ -q_y & q_z & q_\omega & q_x \\ q_x & -q_\omega & q_z & q_y \\ -q_\omega & -q_x & -q_y & q_z \end{bmatrix} \begin{bmatrix} p_\omega \\ p_x \\ p_y \\ p_z \end{bmatrix} \tag{4.25}$$

Euler Angle to Quaternion

To get the quaternion out of an Euler angle, some sine and cosine equations have to be computed. In fact, the quaternion represents in its real value the cosine of the angle and in one of its imaginary parts its sine.

As it can be seen from Equation 4.26, the real part, ω, of the quaternion is the cosine of the rotation angle divided by two and the imaginary part is the vector, $\vec{u}$, times the sine of the rotation angle, α, divided by two. This means that every rotation can be represented by this equation, whereas (q_x, q_y, q_z) define the rotation about each axis.

$$q = q_\omega + i \cdot q_x + j \cdot q_y + k \cdot q_z = q_\omega + (q_x, q_y, q_z) = \cos(\frac{\alpha}{2}) + \vec{u}\sin(\frac{\alpha}{2}) \tag{4.26}$$

Quaternion to Euler Angle

To convert an Euler Angle to a Quaternion is quite easy, however to get back the Euler angle some more difficult equations must be solved. The problem is that in order to get back to the angle representation an arc tangent (atan) has to be solved, which is only defined from $[-\frac{\pi}{2}, \frac{\pi}{2}]$. To solve this problem the atan2 function has to be used, where a case decision has to be made to get it defined from $[-\pi,\pi]$. This function is already implemented in most standard C++ math libraries and we only have to apply it.

$$\begin{aligned} \text{heading} &= \text{atan2}(2 \cdot q_y \cdot q_\omega - 2 \cdot q_x \cdot q_z, 1 - 2 \cdot q_y^2 - 2 \cdot q_z^2) && (4.27) \\ \text{attitude} &= \text{asin}(2 \cdot q_x \cdot q_y + 2 \cdot q_z \cdot q_\omega) && (4.28) \\ \text{bank} &= \text{atan2}(2 \cdot q_x \cdot q_\omega - 2 \cdot q_y \cdot q_z, 1 - 2 \cdot q_x^2 - 2 \cdot q_z^2) && (4.29) \end{aligned}$$

4.3.1 Quaternion-Based Kalman Filter

The quaternion-based Kalman filter is useful for two reasons. First, it reduces the computation time of the Kalman filter. Secondly, it also removes the drift of the gyroscope and allows computation of the angular orientation in a three dimensional space. As described in Section 2.2 the sportSemble system has three different classes of sensors to detect the player's movement at each node. In the following section, the fusion of the gyroscope data and the digital compass data is explained.

The gyroscope measures the angular velocity of a given body segment. Therefore, given an initial orientation, the angular velocity can be integrated over time. This allows us to get the orientation in three dimensional space over time. The main problem in this case is to define the initial orientation, which is only possible here with the compass readings. Therefore, the compass gives the initial orientation and the gyroscopes give the change of this orientation, as the compass can only be sampled at only 6 Hz hence can not provide data at the high rate desired.

The problem with this approach is that the integration over the angular velocity of the gyroscope produces a drift error. This small error for every time step can produce a significant integrated drift away form the real angular position, as it was illustrated in Section 2.2.5. Therefore, it is necessary to fuse the gyroscope data with the data of the compass. The compass only allows sampling every 150 ms, therefore the gyroscope drift can be corrected every 150 ms.

The most important formula to fuse the gyroscope data and the quaternion measurement stats that the multiplication of the quaternion and the angular velocity (Equation 4.30) gives the deviation of the quaternion [JCKC92].

$$\dot{q} = \frac{1}{2} (q \otimes \omega) = \Omega(\omega) q \tag{4.30}$$

As the quaternion multiplication has been defined in Equation 4.25, it is possible to solve

that equation and to get the matrix $\Omega(\omega)$ (Equation 4.31).

$$\Omega(\omega) = \frac{1}{2}\begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ \omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix} \tag{4.31}$$

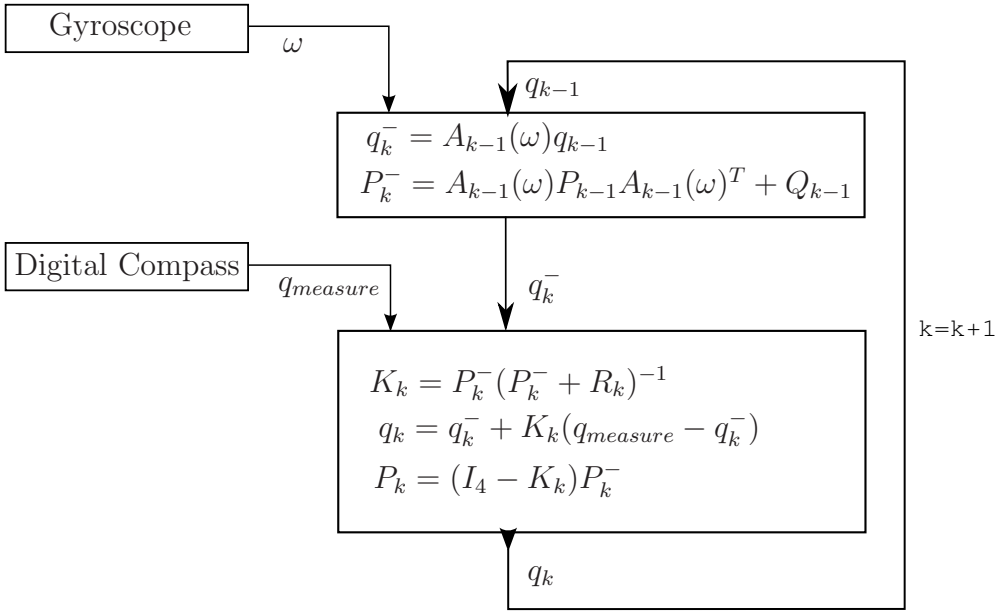

Figure 4.9: Algorithm structure of Quaternion Kalman Filter

From this matrix, $\Omega(\omega)$, we have to get to the state time evolution matrix, A, (Equation 4.13) of our Kalman filter. What we have to do first is to discretize Equation 4.30. We know that the mathematical definition of a first order derivation is (Equation 4.32)

$$\dot{f}(t) = \lim_{\Delta t \to 0} \frac{f(t + \Delta t) - f(t)}{\Delta t} \tag{4.32}$$

$\dot{f}(t)$	Derivation of the function $f(t)$ over time
$f(t + \Delta t)$	The function $f(t)$ one time step later
$f(t)$	The function f depending on the time
Δt	Time step

When we apply this equation to the differential quaternion Equation 4.30, we get Equation 4.33.

$$\frac{q_k - q_{k-1}}{\Delta t} = \Omega(\omega)q_{k-1} \tag{4.33}$$

q_k	Current quaternion state (node orientation)
q_{k-1}	Previous quaternion state (node orientation)
Δt	Time step
$\Omega(\omega)$	Angular velocity matrix

This leads us to a formula to get the state time evolution matrix, A_{k-1}, of our Kalman filter (Equation 4.19). The only thing left to do is to rewrite the equation above.

$$q_k = \underbrace{(\Omega(\omega)\Delta t + 1)}_{A_{k-1}} q_{k-1} \tag{4.34}$$

1	Identity matrix
q_k	Current quaternion state (node orientation)
q_{k-1}	Previous quaternion state (node orientation)
Δt	Time step
$\Omega(\omega)$	Angular velocity matrix (sensor data from gyroscope)

We finally get the matrix, A_{k-1}, which represents the time update of the Kalman filter. In Equation 4.35 the matrix, A_{k-1}, is written in matrix form.

$$A = \begin{bmatrix} 1 & -\frac{\Delta t\omega_z}{2} & \frac{\Delta t\omega_y}{2} & \frac{\Delta t\omega_x}{2} \\ \frac{\Delta t\omega_z}{2} & 1 & -\frac{\Delta t\omega_x}{2} & \frac{\Delta t\omega_y}{2} \\ -\frac{\Delta t\omega_y}{2} & \frac{\Delta t\omega_x}{2} & 1 & \frac{\Delta t\omega_z}{2} \\ \frac{\Delta t\omega_x}{2} & \frac{\Delta t\omega_y}{2} & \frac{\Delta t\omega_y}{2} & 1 \end{bmatrix} \tag{4.35}$$

At this point, it has to be noted that this matrix is normally a constant, as the Kalman filter is conceived to filter out Gaussian distributed errors. In our case, the matrix is updated for every time step by the new angular velocity measured with the gyroscope. Therefore, it is possible to use the Kalman filter for reduction of the drift of the gyroscope by combining it with the quaternion operator of the digital compass. What has to be mentioned here is that the vectors q_{k-1} and q_k have to be normalized between the steps, as the manipulations of q do not require the unit property of the quaternion.

We get the process noise covariance, Q, and the measurement noise covariance, R, the same way as for the general case of the Kalman filter design (Equation 4.18). The only thing necessary to change is that the process noise covariance has to be transformed the same way as the matrix A. In our case, the process noise, w_{k-1}, from the global equations of the time update of the Kalman filter (Equation 4.19) has to be transformed.

$$y_G = \omega + n_G \tag{4.36}$$

Having the gyroscope output defined as in Equation 4.36 with the angular velocity, ω, and the measurement noise, n_G, we still need to perform a transformation operation (Equation 4.37) of the measurement noise, n_G, as the angular velocity gets inserted in the Equation 4.19 using Equation 4.30.

$$w_{k-1} = -\left(\frac{\Delta t}{2}\right) \Xi_{k-1} n_{G,k-1} \tag{4.37}$$

w_{k-1}	Process noise
Δt	Time step
Ξ_{k-1}	Transformation matrix
$n_{G,k-1}$	Measurement noise

y_G	Sensor signal gyroscope
ω	Angular velocity

As Q is the covariance matrix of the process noise and as the noise of the gyroscope and the noise of the digital compass are uncorrelated, the process noise covariance is equal to the square of the process noise vector (Equation 4.38).

$$Q_{k-1} = E[w_{k-1} w_{k-1} T] = \left(\frac{\Delta t}{2}\right)^2 \Xi_{k-1} \sum_G \Xi_{k-1}^T \tag{4.38}$$

Q_{k-1}	Noise covariance of the gyroscope
$\sum_G$	Covariance matrix of the gyroscope measurement noise

$$\Xi = \begin{bmatrix} q_z & -q_y & q_x \\ q_y & q_z & -q_\omega \\ -q_x & q_\omega & q_z \\ -q_\omega & -q_x & -q_y \end{bmatrix} \tag{4.39}$$

The transformation Ξ matrix can be written as in Equation 4.39, and the covariance matrix of the gyroscope measurement noise $\sum_G$ is the same as the autocovariance matrix defined in the global Kalman filter equations in Equation 4.18.

4.3.2 Conclusion on the Quaternion Based Kalman Filter

In the last two Sections 4.2 and 4.3, the mathematical background is explained. The Kalman Filter section describes the global equations of a Kalman filter and explains a small application. The subsequent section about quaternions started also about general quaternion math, but then integrated the quaternion math with the Kalman filter.

The resulting mathematical framework permits us to fusion the compass data and the gyroscope data together. It further allows us to correct the drift in the integrated gyroscope data.

4.4 Merging High and Low Range Sensor Data

As there are two ranges of inertial sensors on each node, it is necessary to use data from the appropriate sensor type, low or high range, for each data point. The algorithm developed to do this is quite simple. The low range sensor gives precise sensor information for the slow movements and the high range sensor system for the fast movements. The idea is to define an area of interpolation and hand off between the two sensor systems before the low range sensor system goes into saturation. This gives us a really good relative precision over the whole range from zero to 120 g. The exact same procedure is used to merge the low and high range data of the gyroscope together.

In the following code example (code example below), the low and high range accelerometer data is merged. The interpolation is made within the last 10% of the low range. The vector *acc_low* represents the low range data and the vector *acc_high* represents the high range data. The constant G is the gravity of the earth defined as $9.81\frac{\mathrm{m}}{\mathrm{s}}$. The variable *dist2max* is the distance between the actual low range value and the saturation limit of the sensor. This value represents the weight used on the value of each vector. If the low range acceleration value is below the 90% of the saturation value, the acceleration chosen is just the value of the low range sensor.

```
for(int i=0;i<3;i++) //over all axes
{
  if (acc_low(i) > 6*G)
  {
    res(i) = acc_high(i);
  }
  else if (acc_low(i) > 5.4*G) //if it is just one axis
  {
    dist2max = (6.0*G - acc_low(i))/6.0/G*10;
    res(i) = acc_low(i)*dist2max + acc_high(i)*(1 - dist2max);
  }
  else
    res(i) = acc_low(i);
}
```

As the same method is been used for the interpolation of the low and high range gyroscope sensors, the gyro interpolation code will not be presented here. The limit for the start of interpolation is set the same way as for the accelerometer, at 90% of the low-rate saturation limit, which means $270\frac{\mathrm{deg}}{s}$ for the gyroscopes.

At this point it is important to note how the different sampling rates of the two sensors are fit together . As the low range sensors are only sampled every 3 ms and the high range sensors every 1 ms, the low range sensor data has to be stretched out. This stretching was done by a weighted distance between the data points. As the low range sensors are sampled with 333 Hz, the high range sensors deliver 3 times as many data points (sampled with 1000 Hz). In order to have the same vector length for the two different ranges, it is necessary to stretch the low range vector by two additional datapoints. The distance of these interpolated datapoints is used as weight, because this allows to give a better repartition. The first interpolated point has on one side the distance of $\frac{1}{3}$ to the previous data point and a distance of $\frac{2}{3}$ to the following data point. Therefore, the weight of the interpolation is the inverse of this distance, as the interpolated point is closer to the previous datapoint. This means that a datapoint that is only one sample time away from the interpolated point gets multiplied by the factor of $\frac{2}{3}$, and the data point with a distance of two samples is scaled with the factor of $\frac{1}{3}$. The same procedure is used for the second datapoint, where here the distances are the other way around. It has to be noted, that this approach is not the best approach, as it is just a linear interpolation. A polynomial interpolation would be more precise as it uses more than just the two nearest datapoints.

The code example below shows the stretching of the low range data vector. The first values of the vectors (*Lo_X*, *Lo_Y*, ...*Lo_Yaw*), representing the low range sensor data of the accelerometer and gyroscope, get populated. In the following two steps, the values in between get computed by this weighting of the measured values. Step one computes

the first datapoint (line 13 to 21 in the code example). Step two computes the second datapoint (line 23 to 31).

```
Low_Gesture=DB->CALIB_LOW_GESTURE_DATA(seq_id ,NODE_LOCATION);
i=0;
while(Low_Gesture.fetch(row))
{
  Lo_X[i] = atof( row[8].data() )*9.81;
  Lo_Y[i] = atof( row[7].data() )*9.81;
  Lo_Z[i] = -atof( row[6].data() )*9.81;
  Lo_Pitch[i] = deg2rad( atof(row[11].data() ) );
  Lo_Roll[i] = deg2rad( atof(row[10].data() ) );
  Lo_Yaw[i] = -deg2rad( atof(row[9].data() ) );
  i++;
  //the two interpolated values
  if(i>3)
  {
    Lo_X[i-3] = (Lo_X[i-1] + 2.0*Lo_X[i-4])/3.0;
    Lo_Y[i-3] = (Lo_Y[i-1] + 2.0*Lo_Y[i-4])/3.0;
    Lo_Z[i-3] = (Lo_Z[i-1] + 2.0*Lo_Z[i-4])/3.0;
    Lo_Pitch[i-3] = (Lo_Pitch[i-1] + 2.0*Lo_Pitch[i-4])/3.0;
    Lo_Roll[i-3] = (Lo_Roll[i-1] + 2.0*Lo_Roll[i-4])/3.0;
    Lo_Yaw[i-3] = (Lo_Yaw[i-1] + 2.0*Lo_Yaw[i-4])/3.0;
  }
  i++;
  if(i>3)
  {
    Lo_X[i-3] = (2.0*Lo_X[i-2] + Lo_X[i-5])/3.0;
    Lo_Y[i-3] = (2.0*Lo_Y[i-2] + Lo_Y[i-5])/3.0;
    Lo_Z[i-3] = (2.0*Lo_Z[i-2] + Lo_Z[i-5])/3.0;
    Lo_Pitch[i-3] = (2.0*Lo_Pitch[i-2] + Lo_Pitch[i-5])/3.0;
    Lo_Roll[i-3] = (2.0*Lo_Roll[i-2] + Lo_Roll[i-5])/3.0;
    Lo_Yaw[i-3] = (2.0*Lo_Yaw[i-2] + Lo_Yaw[i-5])/3.0;
  }
  i++;
}
```

5. Results

In the last chapter, a method of inertial sensor data fusion using a physics engine was presented. It is now possible to compare the accelerations and linear velocities that the system outputs to a camera based motion tracking system. As the inertial sensors directly measure the accelerations and angular velocities, the resultant forces and torques are believed to be more precise than the result of a camera based system.

First, some simple validations will be presented in the context of having a collision between physical objects, a force in a joint, and a torque in a joint. Further, a pitch is analyzed using several medically relevant metrics. As a final result, the inertial system is compared with a Vicon optical tracker. At the end of this chapter, the limitations of the system are discussed.

5.1 Validation

For the validation of the force and torque results, some simple cases are simulated. In these cases, gravity is set to zero in order not to influence the simulations. These validations are important, as it is necessary to know how precise the physics engine simulates the different cases. The cases are constructed as simply as possible, in order to be able to compare their result with a manually calculated result. They are all similar to different situations that occur during a real simulation. First, the force of a collision is compared, then the force within a defined joint and finally the torque within a defined joint is evaluated.

5.1.1 Validation of the Force of a Collision

To validate the simulation, a simple collision of two objects has been programmed. Two boxes, each having the same fixed velocity, mass and impulse, hit each other. Figure 5.1 shows the two boxes. This validation shows the results of moving objects that collide with each other without having a defined physical connection between the two.

This should produce a force depending on the impulse force transformation after the

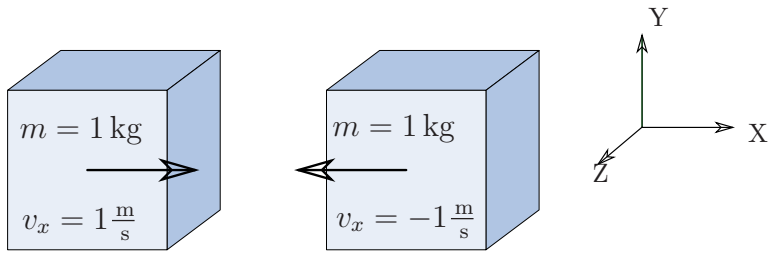

Figure 5.1: Validation of the Impulse Force Calculation

impact of the two equal boxes.

$$F = m \cdot \frac{\Delta v}{\Delta t} = \frac{\Delta I}{\Delta t} = \frac{I_{t-1} - I_t}{timestep} = \frac{2\,\mathrm{J}}{1\,\mathrm{ms}} = 2000\,\mathrm{N} \tag{5.1}$$

F	Force
m	Mass
Δv	Difference of velocity
Δt	Difference of time
ΔI	Difference of impulse
I_{t-1}, I_t	Impulse at one time step before and at this time step
time step	Time step of the simulation

Figure 5.2 shows the result of this basic collision. As can be seen from the two graphs, the impulse of the two boxes goes immediately to zero and the force between these two boxes goes up to 2000 N (Equation 5.1) for one millisecond. This example validates the

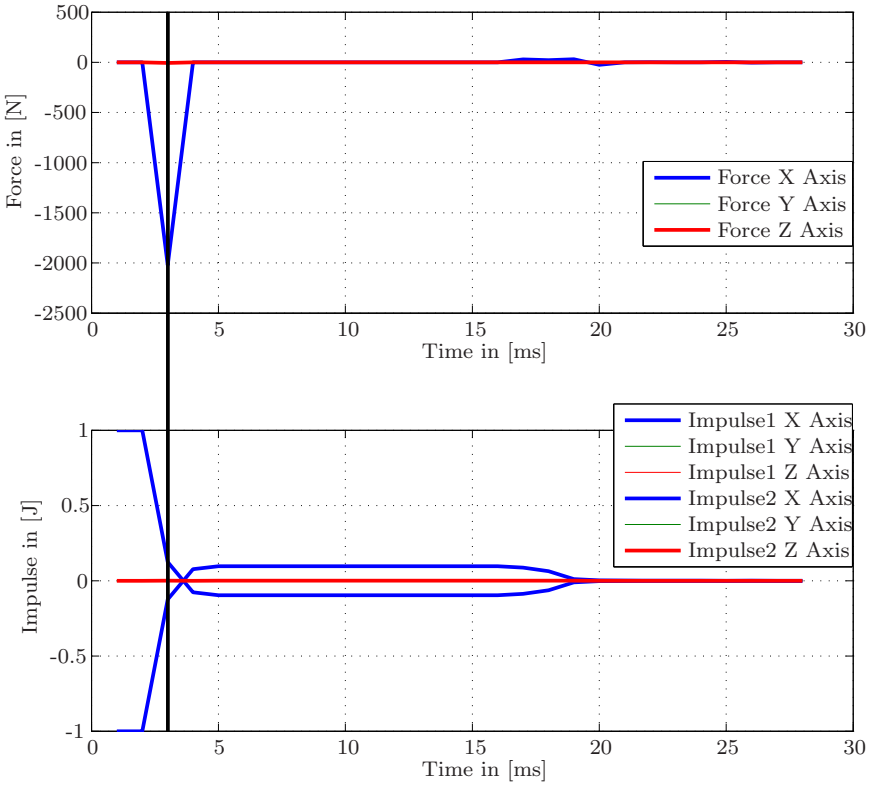

Figure 5.2: Graph of the Impulse and Force of the basic collision

force calculation in the physics engine for a collision, but it does not validate the behavior of the impulses and forces in the joints. Between 5 and 20 ms an error in the impulse can

be seen. This error is the result of a not perfect collision of the two boxes. Both boxes rest with a small impulse instead having no impulse at all. This is due to the solver modeling a stiff spring in between of the boxes instead of simulating a perfect collision.

The collision represents a different situation than solving the kinetics of joint-related objects. In this case, the solver did not know that the two objects would be in contact with each other unless they were already colliding. The solver must figure out when the two boxes actually hit each other. The physics engine used in this simulation [cCTRLtH09] proposes two solutions for a collision. The first solution is the discrete solution, where the collisions of objects are only detected at discrete time steps. That means that a fast object could eventually penetrate another object before the collision is detected. This method is not the most accurate method, but allows fast computation of collisions. The other method employs a so-called continuous time simulation. That means that if the objects will collide over a time step, an additional step will be added. To avoid the collision check for every object in the world (problem size of $O(n^2)$), the objects are separated into narrow and broad groupings. This allows a much faster calculation for a large amount of objects.

5.1.2 Validation of the Force in a Joint

The force in a joint between two boxes is examined in this section by applying a known force on one box and by calculating the force that will be created in the joint between the two boxes. This case differs from the one illustrated above, as there is now a joint constraint defined between the two boxes. This constraint does not allow the second box to intrude into the first box at all (in the case of a collision, there can be a some error if the first box intrudes into the second box).

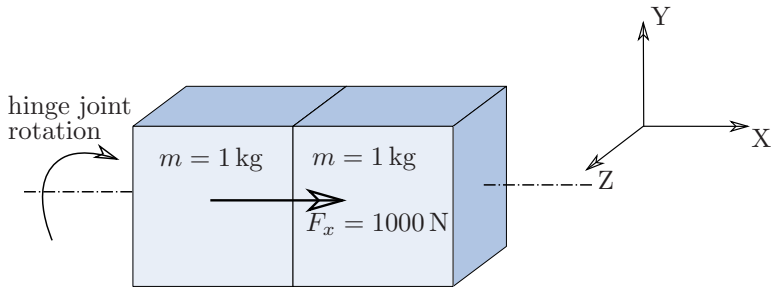

Figure 5.3: Boxes Having a Hinge Joint Relation

The two boxes, which can be seen on Figure 5.3, have a hinge joint defined between them. It allows them to turn about the x-axis (displayed by a dashed line), which is the only rotation possible without having one box intruding into the other one. The calculation is the same as in the previous section (Equation 5.1). The result of this calculation provides better results than for the collision. This is due to the definition of the joint between the boxes, because the engine already knows that the two boxes are touching each other at the pivot point of the joint. Therefore, no penetration of the boxes is possible.

A force, F_x, of 1000 N was applied to just one box, and only for a single 1 ms time step. Therefore the resulting impulse should give 1 J for one box (Equation 5.2), but as they are connected and have the same mass, the impulse at each box should be half.

$$I = \int_0^T F \, \mathrm{dt} = \int_0^{1\,\mathrm{ms}} 1000\,\mathrm{N} \, \mathrm{dt} = 1000\,\mathrm{N} \cdot 1\,\mathrm{ms} = 1\,\mathrm{J} \tag{5.2}$$

In Figure 5.4 the results of this simulation are depicted. At the time of three milliseconds, a force of 1000 N is applied, which lasts only for one millisecond. This force first created an impulse of 1 J at the box where it is applied but than was halved as there is the second box attached. This impulse then lasts for both boxes, as there is no external force.

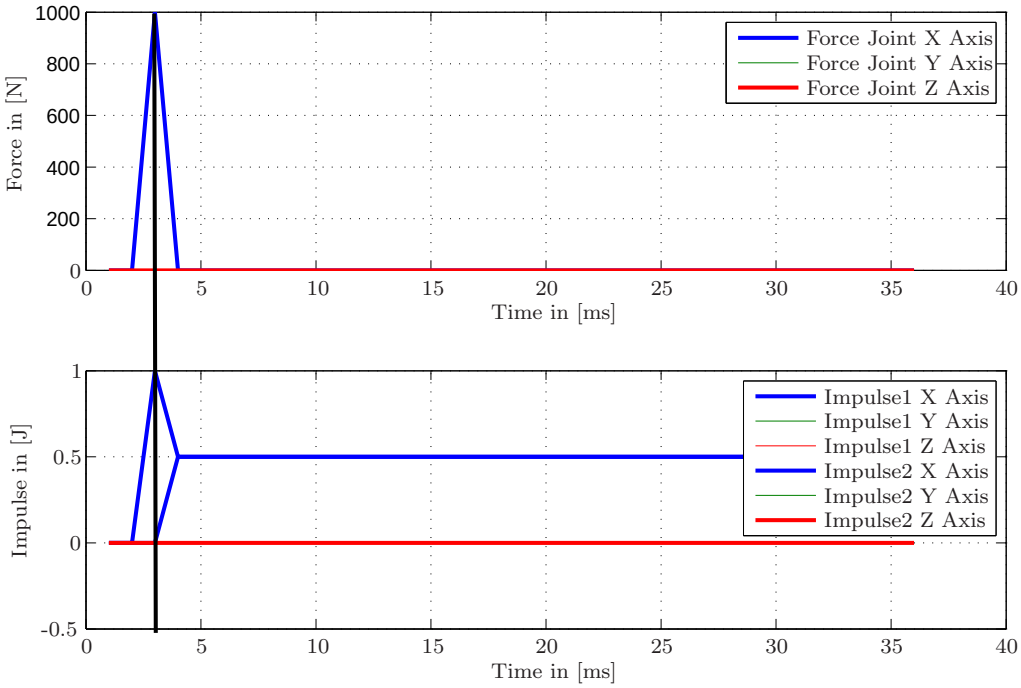

Figure 5.4: Joint Force and Impulse of the two Boxes

5.1.3 Validation of the Torque in a Joint

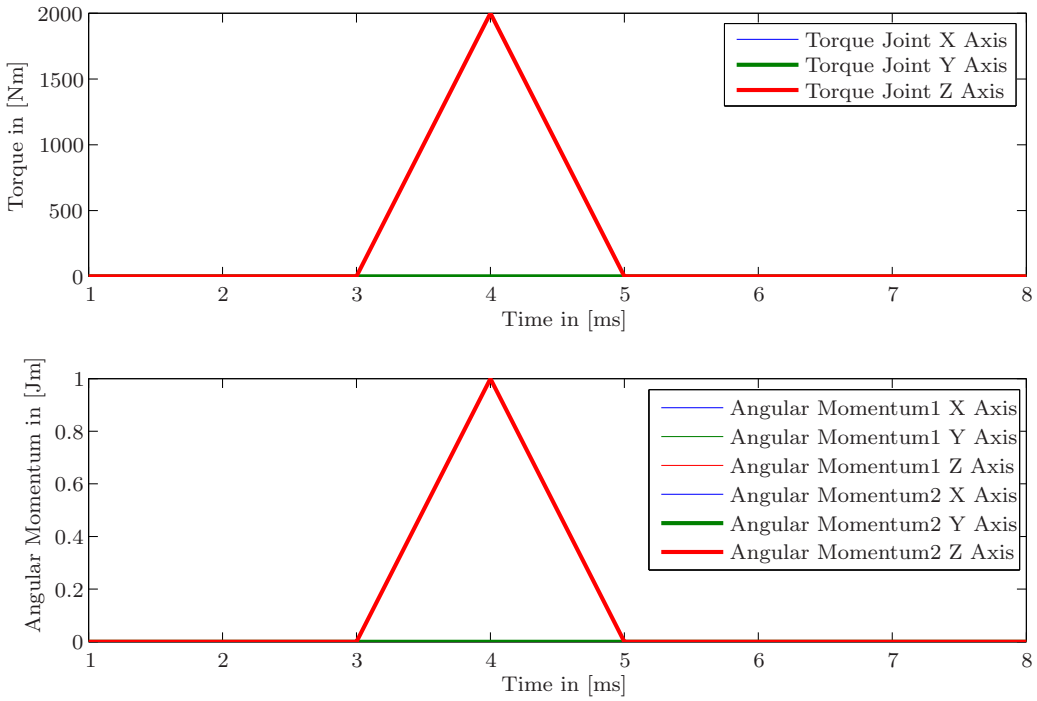

Figure 5.5: Validation of the Torque in the Joint

As explained in Section 4.1.6, the forces and torques are computed using the change in linear and angular velocities before the solver step. In this validation test, the two boxes again have a hinge joint (displayed by a dashed line) defined between each other and are in the configuration as shown in Figure 5.6.

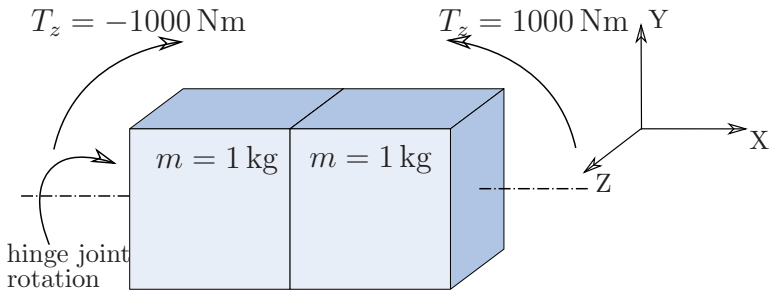

Figure 5.6: Boxes Having a Hinge Joint Relation

The difference is that here a torque will be applied on both of the boxes, but not in the direction the joint can turn. The torques are both applied around the z-axis, and the joint restricts movement to only the x-axis. The torques are chosen to have the same value, but

in inverse directions in order to create a torque between the boxes, while not moving the boxes at all.

The result of the simulation are exactly what the ideal mathematical calculation gives. On both boxes, a torque, T_z, of $\pm 1000\,\text{Nm}$ is applied. This gives a torque of 2000 Nm in the joint, but no movement at all, as they are both applied for 1 ms and in inverse directions.

Equation 5.3 shows how the torque is computed. The physics engine is of great help in this case, as it provides the matrix of inertia for each element. Then the torque can be computed by first order differentiation of the angular momentum.

$$T = \frac{d}{dt}(L) = J \cdot \dot{\omega} = J \cdot \frac{\omega_{t-1} - \omega_t}{timestep} \tag{5.3}$$

T	Torque
J	Matrix of Inertia from the center of the joint
$\dot{\omega}$	Angular velocity derived by the time
ω_{t-1}, ω_t	Angular velocity at time $t-1$ ant at time t
time step	Time step of the simulation

5.2 Results of a Pitch

In the following subsections, the impulse, force and torque within the joints of a human body are presented. This test crosses over between low and high range sensor data at an acceleration of 5.4 times gravity and 270 degree per second for angular velocity. The interpolation of the two sensors is currently done in a linear way (Section 4.4). All simulation results are in body coordinates to be able to interpret the results easily.

5.2.1 Angles in the Joints

As the whole movement is reconstructed physically, the joint angles can be exported into segment coordinates, such that they can be easily interpreted.

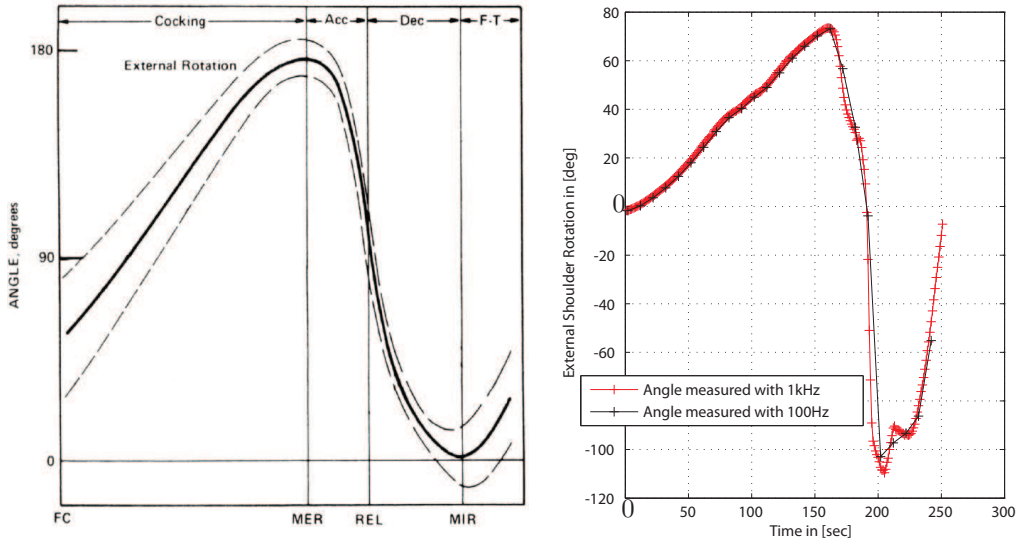

Figure 5.7: External Shoulder Rotation during Pitch

The example of the joint angles shown in Figure 5.7 is the external shoulder rotation. As this rotation is the most important rotation in accelerating the ball, this joint angle is presented here. On the left, the joint angle of professional pitchers (optically measured) is taken from [FG96], and on the right a pitch by the author is displayed. Our data is presented with a 1000 Hz sampling rate, as well as how it would look like with a 100 Hz sampling rate (the 100 Hz sampling rate corresponds to the velocity of a medium budget camera system). The slower data stream is derived via a moving average filter with a window of 10 from operating on the 1000 Hz sampled data. One obvious difference between the two graphs is that the graph on the left has an offset of 90 deg. This is due to a difference in the definition of the origin in the two systems.

Two observations can be made from these plots of the external shoulder angle. By comparing the image on the left and the image on the right, it can be observed that

the professional pitcher can rotate his arm further back than the author. This gives the professional pitcher a longer distance to accelerate the ball. Therefore, as expected, the author won't get a pitching speed as fast as that of the professional pitcher. The second observation can be made by comparing the measurements sampled at 1 kHz and those sampled with only 100 Hz. The red curve with 1 kHz sampling rate has two sharp bends and a few other sharp features, which would not be observed with a camera system having only 100 Hz sampling frequency. Although more needs to be done to validate the biomechanical accuracy of these curves (the shoulder is indeed a complex joint, and measuring external points not attached to the bone is a natural source of error - e.g., the IMU board could snap across its mounts during a fast gesture), these sharp bends could be interesting, as they might indicate fast dynamics happening in the shoulder during this moment. The data of the 1 kHz example is not filtered and directly represents the output of the sensor system.

5.2.2 Impulses in the Joints

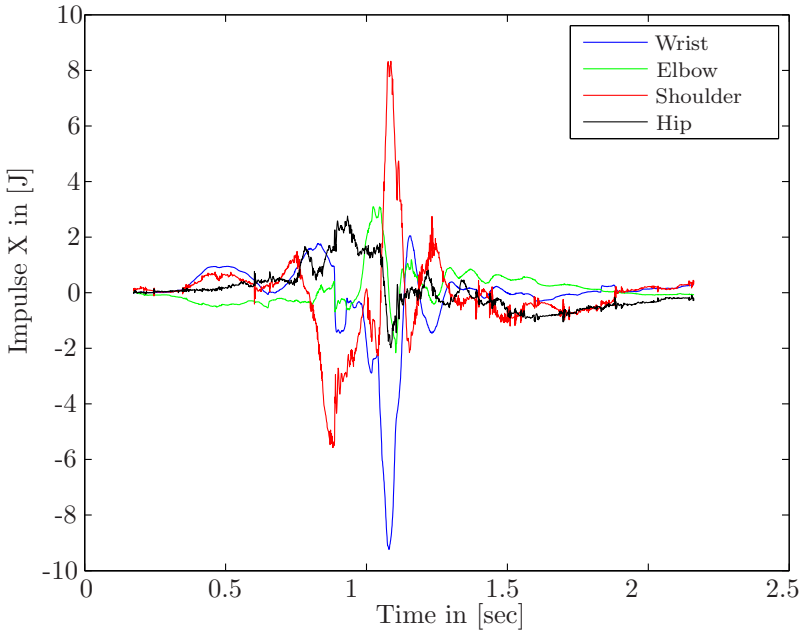

Figure 5.8: Impulse of a Pitch by the Author

The impulse, also called linear momentum, in the joint can be obtained as a result of the physical simulation using the inverse kinematics solver of the physics engine. The impulse directly represents the integration of the force over time. It is the product of the force and the amount of time it is applied. As a result, an impulse may also be regarded as the change in momentum of an object to which a force is applied. However, it provides a

useful model for computing the effects of ideal collisions. As this is the main parameter the physics engine is using, it is interesting to examine it. Furthermore the impulse results are used to calculate the forces in the joints by using the first derivative of the impulse. Impulse represents the mass of the body times its velocity. This means that if there is a large change in the impulse of a body, there is a large force on this body.

Figure 5.8 shows the impulses of a pitch. Interesting in this case is that the impulse of the wrist is almost equal to the impulse of the shoulder. As the impulse in the joint is the difference of the velocity of the two attached segments (hand and forearm) multiplied by the mass of the segments, it can be concluded, that the difference in velocity of the wrist segments is much larger than the difference in the velocity of the shoulder segments.

5.2.3 Forces in the Joints

The forces in the joints are the derivative of the impulse of the joint. This means that all of the changes in the impulse are scaled by the sample frequency. This would give a relatively noisy output if the differential solver was involved in the process. However, if the forces are calculated using the method described in Section 4.1.6, the force output does not need to be filtered to be interpretable.

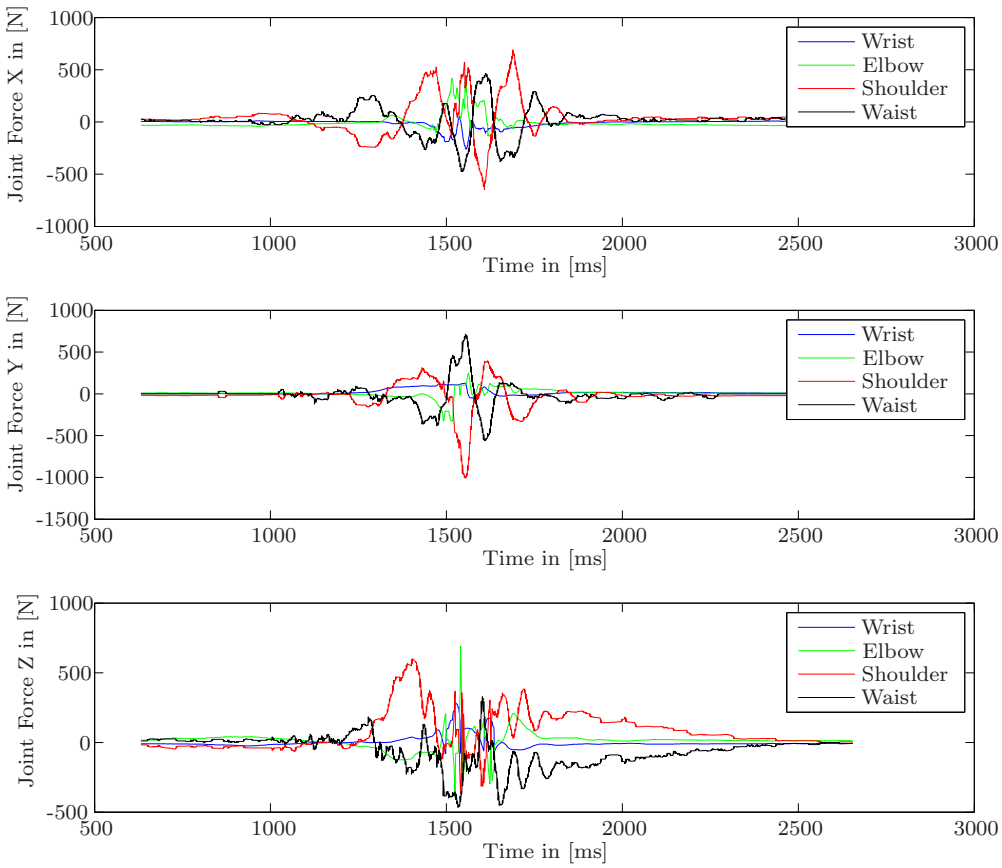

Figure 5.9: Forces of all Joints in X, Y and Z of a Pitch of the Author

Figure 5.9 shows the forces of all upper body joints during a pitch by the author. The three graphs are separated in the X, Y and Z directions. The x-axis is to the left of the body, the y-axis is towards the sky and the z-axis is pointing out front. All these three directions are in body coordinates. This means, that the inertial direction of the axes changes with the segment orientation. It has to be noted here, that the force in the upper arm and the waist oscillates in the Z direction. This could be the result of a flopping node on the upper body and hip, which still can not be fixed in an adequate way.

Figure 5.10 shows the published shoulder compression force of a professional pitcher on the left [FG96] and the simulation results of the author pitching on the right.

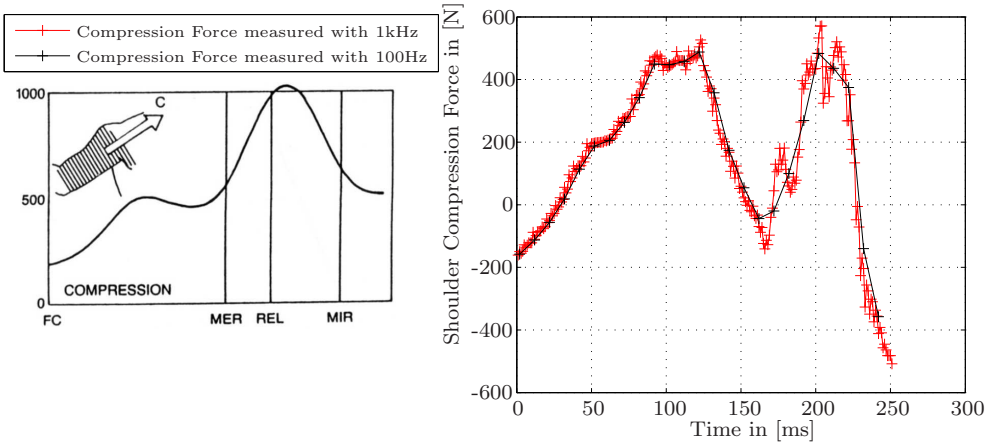

Figure 5.10: Compression Force Compared to Paper [FG96]
Left: Compression Force of a Professional Pitcher
Right: Compression Force of the Author Pitching

The difference between the two pitches is that the author did not go one step forward to pitch, and therefore does not have as high of a compression in the shoulder as the professional pitcher does. Also, the dual peak is somewhat different, as the author does not have the force to get up to 1000 N for the shoulder compression force. Again, this second peak force is due to the trained step forward movement and comes therefore from the lower body segments. In red, the result of the system using 1 kHz sampling frequency is presented and in black the same data only having a sample frequency of 100 Hz. To sample down from 1 kHz to 100 Hz again a moving average filter is used, having a window size of 10. Clearly the higher frequency data shows significant features that are not seen in the lower frequency data - again, this may be due to mounting artifacts and not biomechanics, although things like this need to be explored further.

5.2.4 Torques in the Joints

Medically, the most interesting results of these physical models may be the torques occurring in the joints. These represent critical mechanical information, as torque is basically responsible for breaking a joint. As it is interesting to have the torque in body coordinates, the torque is presented in the body frame, as it was done with the force and the joint angle.

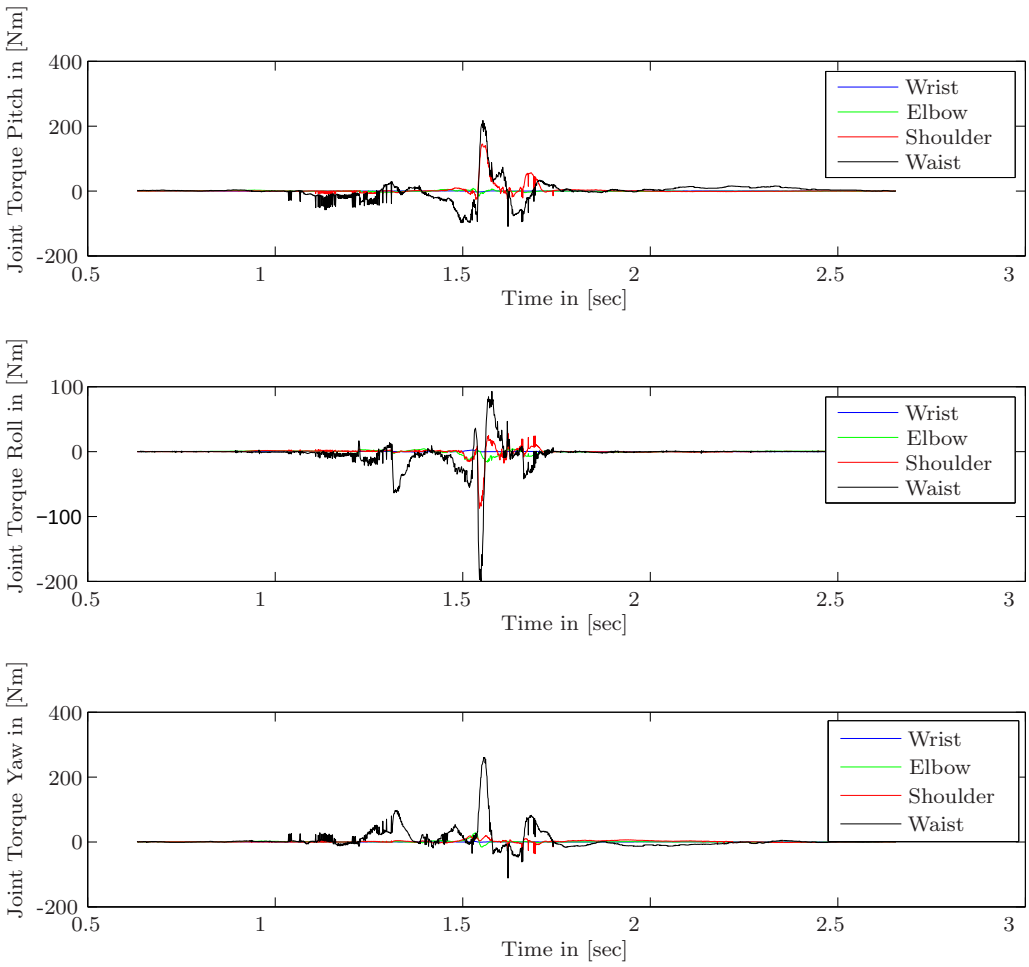

Figure 5.11: Torques of all Joints in Pitch, Roll and Yaw

Figure 5.11 depicts the torques of all joints of the upper body during a pitch. The first graph show the pitch, the second the roll and the third the yaw rotation of the torque in body coordinates. Quite suspicious seems that the highest torque is at the waist, which is not moving very fast. But the mechanical explanation of this phenomena is quite simple, because the waist is the joint where all the torques are added.

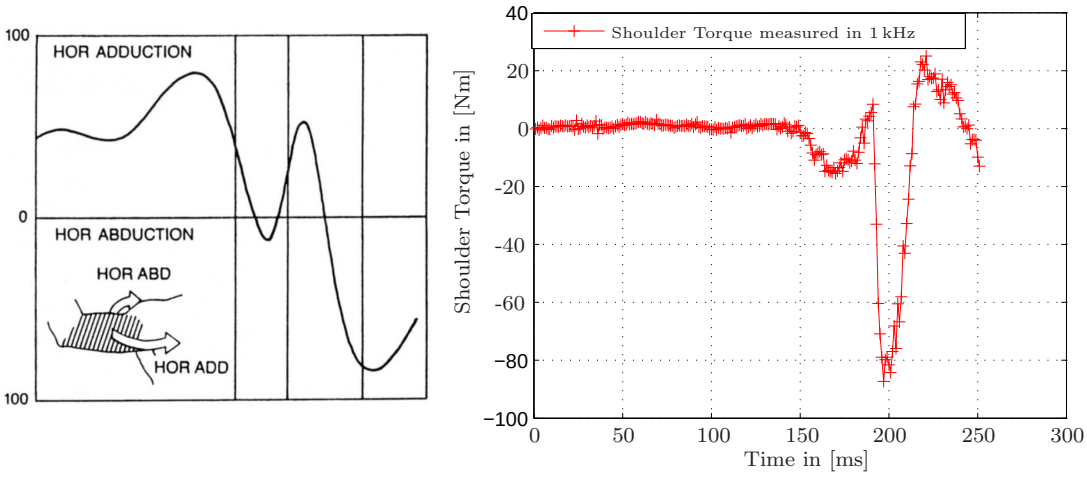

Figure 5.12: Shoulder Torque Compared to Paper [FG96]
Left: Shoulder Torque of a Professional Pitcher
Right: Shoulder Torque of the Author Pitching

Figure 5.12 shows the unfiltered torque of the shoulder joint during a pitch. As the results are visible without filtering, no filtering algorithms have been applied.

While the simulation results and the result from the book [FG96] are somewhat different, they both show a high negative torque at the end of the pitch. It also has to be considered that our data is not from a professional pitcher, but from the author himself performing a pitch not using the one step acceleration with his foot. Therefore it is natural that there is no torque at the beginning of the pitch, for example. Further work needs to validate the differences and structures exhibited in this curve.

5.3 Angle compared to an Optical System

As optical motion tracking systems are the current state of the art in motion measurement, it is necessary to compare the inertial measurement system with an optical system.

The optical system used was the Vicon 8i using MCAM M2 cameras. It has a 100 Hz sampling rate and a resolution of 1.3 megapixels per camera. On the top of each inertial node, three optical markers are placed (Figure 5.13). This way, the orientation and placement of the inertial measurement nodes are the same as the orientation and

placement of the optical markers.

This setup allows us to compare the angle that the optical system measures with the angle the inertial system gets by integrating the linear acceleration and angular velocity over the duration of motion. In order to get an accurate angle from the optical system, all

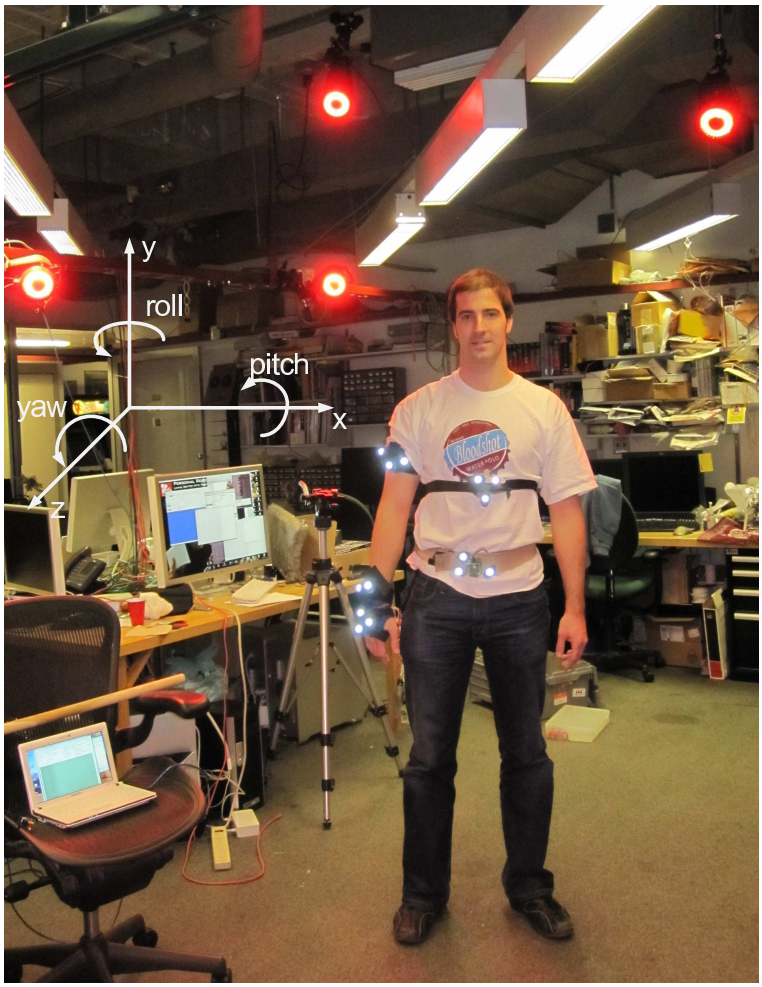

Figure 5.13: Optical Validation with a 100 Hz Vicon System

three markers have to be seen by three cameras. As this is not always the case, the angles may jump. The only way to prevent this from happening is to install as many cameras as possible (the system used has 11 cameras).

Since the shoulder is one of the most interesting joints of a pitcher, the roll angle (see coordinate system in Figure 5.14) of a simple gesture has been compared between the optical and inertial system. The gesture was a movement of the arm around the y-axis (roll) having the arm up and then putting the arm down (yaw).

As you can see from Figure 5.14, the inertial measurement system's roll angle aligns almost perfectly with the optical system, but breaks apart a little just before the end and recovers at the end of the gesture. The angle measured with the inertial system is the

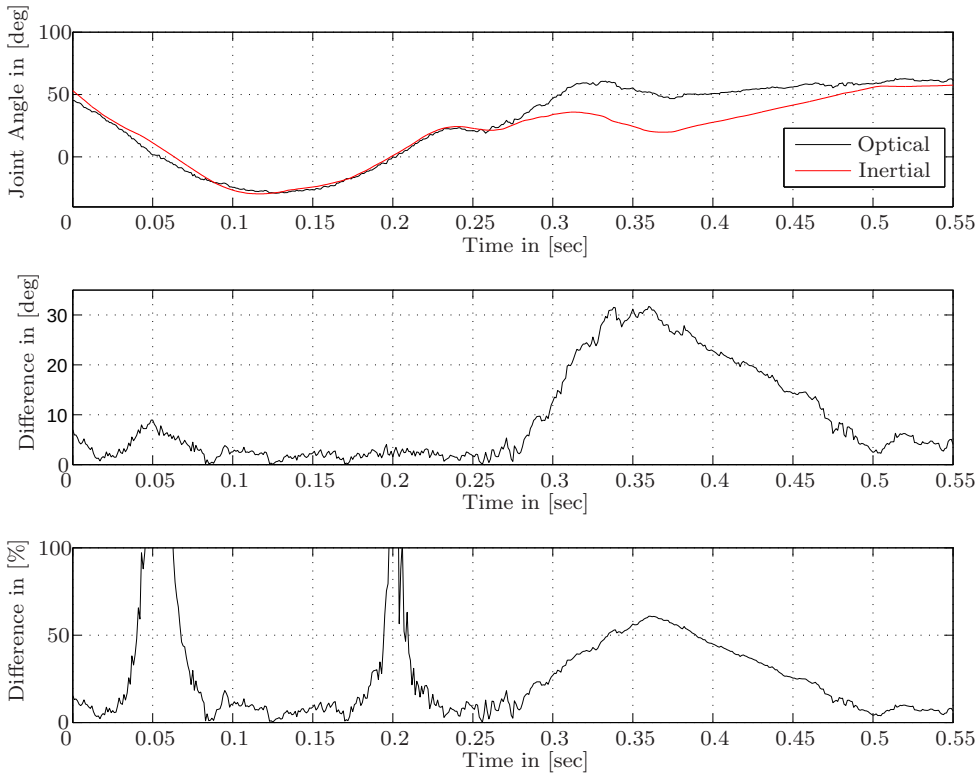

Figure 5.14: Comparison of the Shoulder Roll Angle

result of integration over the linear acceleration and the angular velocity. The interesting point of this comparison is that during this measurement there was just a roll movement, hence the calculated inertial angle basically follows the optical angle (till 0.25 sec). From the time 0.25 sec to 0.3 sec there is an additional yaw movement (arm moves down) in the gesture and during that time the angle measured with the inertial system falls up to 30 deg apart, but recovers after some milliseconds.

In order to understand why the joint angles go 30 deg apart, another calculation has been done leaving out all the accelerometer data and setting the gravity of the physical simulation environment to zero. This way, only the angular velocity measured by the gyroscopes get integrated over time. The result (Figure 5.15) shows a similar problem to that occurring with using the accelerations 5.14. The angle of the inertial system again diverges by about 30 deg. However, the angle does not recover as before with the accelerations applied.

One possible explanation to this behavior could be errors in the node orientation offset explained in Section 5.4.1. The angular movement is precise during the whole roll movement, except when there is an additional yaw movement at the time 0.25 sec. That means that the sensor is rotated in a way that the roll angle measurement is not affected as long

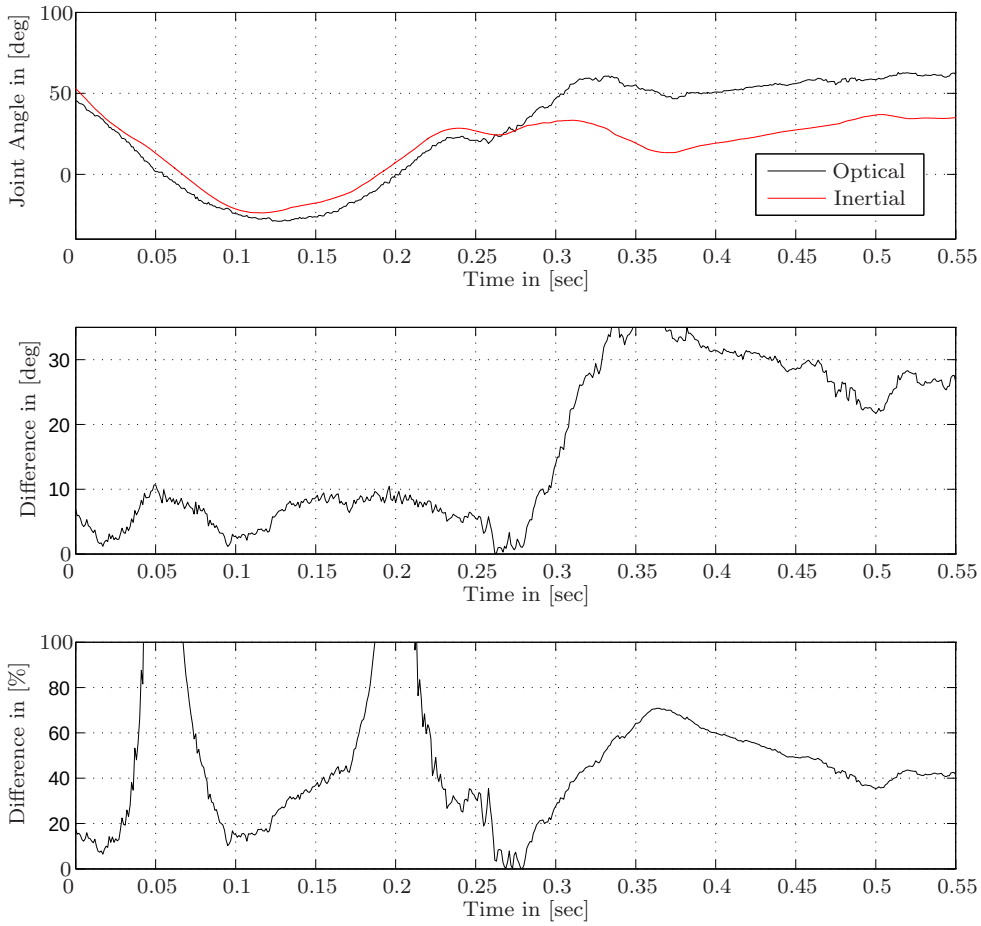

Figure 5.15: Comparison of the Shoulder Roll Angle using only the Angular Velocity

as there is no pitch or yaw movement. The rotation offset of the node towards the segment is discussed in Section 5.4.1.

5.4 System limitations

As with any system, there are factors that limit the precision of the physical model developed. The first of these is the precision of the sensors. Small offsets of the accelerometer over many time steps cause an object to be positioned far from its real position. In the Section 2.2.5, this issue is discussed based on the hardware of the actual system.

A second limitation was in the method for placing the nodes on the body, particularly on the arm. For the forearm and hand node, there were almost no issues, but for the upper

arm node an offset of up to 60 degrees could occur.

Therefore, it is necessary to think about an orientation calibration system, which calibrates out the node orientation with respect to the segment orientation.

Another issue is that the model of the athlete is not complete. It only takes into account the athlete's upper body movements and not the lower body movements. This means we assume that the legs are fixed to the ground and do not move at all, therefore the fact that most pitchers take a step forward to accelerate their body is not taken into account by this system.

5.4.1 Node Segment Orientation Offset

The force and angular velocity driven model has some assumptions built into it about the placement of the nodes on the body and is very sensitive if a node is not attached to the body in the assumed orientation. An improperly oriented node on a segment will cause a drift in a direction that is different than it is moving in reality. This angle, if it is not taken into consideration, will change the accelerations over the three axes as well as the angular velocities. This can create movements that are not physical and will therefore make the solver block the movement. This means that the orientation error will even get bigger, as the impulse of the limited segment will be transferred to other segments.

This orientation error can be significantly reduced by knowing the orientation offset of the node towards the segment. This information can be used to rotate the node back towards its ideal position on the segment. Figure 5.16 shows ideal and real orientation of the nodes on the body of the athlete. The small coordinate systems show the ideal segment orientation (black) in the real coordinate system (red). We propose two methods of getting a known node orientation with respect to the ideal orientation.

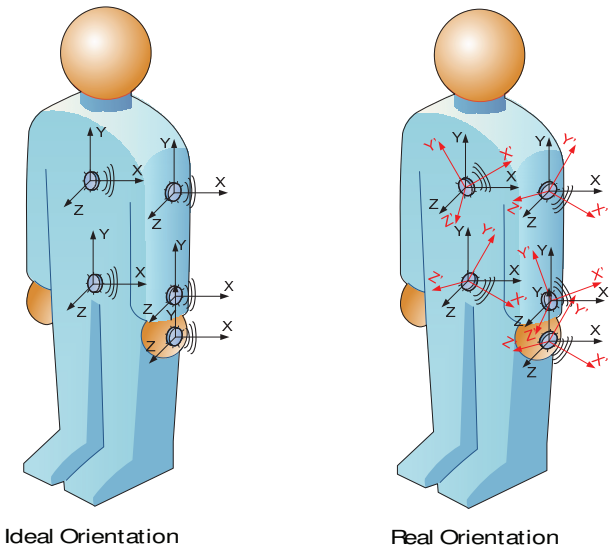

Figure 5.16: Node Orientation Offset

The first is to have the athlete lie down on the ground and have his hands flat on the ground. As the ground is a fixed reference, the differences of the node orientation towards it can be taken into consideration from the compass and accelerometer readings. In this case the athlete has to lie still to provide the most accurate compass reading, as any movement will perturb the compass reading. The second approach is to have the athlete standing with his arm flat at his side against a wall. This technique provides a known orientation of the nodes and can therefore be used for the orientation calibration. Other procedures of this sort that constrain the node orientations for calibration measurement are possible. As our tests with real athletes are often time-constrained, a procedure that minimizes any delay will be preferred.

For the physical model, this orientation can be considered as an offset quaternion for each node. It is trivial to apply such an offset quaternion at each step of the simulation.

6. Conclusions

This thesis presents an approach of fusing the data from a multi-sensor inertial measurement system and compass to provide motion analysis. It explains how to process the data in order to output medically relevant indicators or obtain some idea of the stresses the body undergoes during extreme physical activity. The physics engine *Havok* provides a basic physical body model. By applying measured accelerations and angular velocities on modeled body segments having the correct mass, a new dimension of physical analysis is possible. The physics engine solves the inverse kinematic problem and the framework of the Kalman filter combines the information from the different measurement systems in order to allow for optimal correction. The Kalman filter provides a way to reduce drift during slow motion by updating the angular orientation using data from the gyroscopes and the digital compass.

This approach allows us to derive angles, forces and torques across the joints of an athlete using unfiltered data of the inertial measurement system. These results are all achieved by separating the inverse kinematic solver and the step of applying the inertial sensor data.

6.1 Data Fusion Evaluation

The physical model currently provides valuable force and torque data, but does not provide precise enough joint angles. This angular error should be able to be reduced by storing the magnetometer data of the compass separately. This would provide more information during extreme athletic motion. At the moment, the magnetometer data is combined with on chip accelerometer data within the digital compass. Hence, during high dynamic motion, the acceleration vector does not point towards gravity, and therefore induces errors in the compass reading.

The upsampling of the low range sensor data is done linearly, but much better precision could be obtained using a polynomial interpolation. At the moment only the two proximate data points are considered, but also using more points in a higher-order interpolation could bring more precision.

6.2 High Level Evaluation

The approach that this thesis presents is a valuable new idea of incorporating inertial sensor data into a physical simulation. It allows us to use measurements of accelerations as being proportional to real forces on the segments, and does not use only the gyroscope data during the high athletic motion. Although more investigation is warranted, the forces and torques that we infer across the body joints look to be comparable to optical systems while potentially providing better precision and higher sampling rate.

Using small, infrastructureless, portable sensors like our inertial nodes allows athletes to be measured in their natural environment (e.g., baseball field) and therefore aids in allowing doctors to analyze the athletic motions without having the athlete come to them. This allows a much better symbiosis between doctors and athletes. If the system is used to regularly monitor athletes, doctors will have the ability to forewarn them before an injury develops. No system until now provides such ability of medical management over distance for sports-related activity. Therefore, this system could enable much more effective medical care of professional athletes and could prevent athletes from longterm injuries. Some commercial systems, like the Nikeplus [Nik09] or the Body Bug [Mya09] already provide some very high-level medical information (e.g., calories consumed over exercise), but this inertial analysis provides much more depth in the athletic information.

6.3 Future Work

In the future, the body biomechanics model must be improved. The model consists of basic joints, which do not correspond fully to the joints found in the human body. More complex joints have to be modeled together with a joint correspondent spring damper system, which will allow measurement of the expansion movement of the joint.

The ideal approach would be to have the solver of the inverse kinematics model not to restrict the movements of the body segments using the physics engine's own constraint system (Section 4.1.4). In order to achieve this, the sensor calibration must be improved, and the sensors have to be fixed to the body in a more dependable and repeatable manner. The ideal would be to have an array of sensors to describe the movement of every segment. In this way, the drift of sensors could be reduced, and the sensor position on the segment does not influence the measurement in a very critical way.

Furthermore, it would be interesting to "dive" deeper in the Kalman filtering in order to define a Kalman filter using all sensor data to correct each other. The Kalman filter presented in Section 4.2 opens up the first step to a more complex filter fusing together the data in a more sophisticated way.

To provide some valuable information during an athletic performance, muscular fatigue is an interesting metric to analyze. The system could be developed further to allow for real time data analysis as activity is ongoing. This would allow detection of when an athlete is at the limit of his physical conditioning and to replace him on the field if possible. Although such measurement may not be allowed in professional games, this capability would better inform predictive models that could be independently tailored to each athlete, and run with coarse data gathered manually (or perhaps through computer vision) during a game. This diagnostic capability would also be very valuable during training, because it would prevent athletes from training over their physical limits and "overtraining" themselves.

An interesting validation approach would be to use this system with a well-defined robotic arm, and to compare the forces and torques applied on the robotic joints with the forces and torques measured by the system. In a long-term study, it should be even possible to control the movements of humanoid robots using an inertial measurement system. This

could, for example, replace the optical motion tracking system for production robots.

Our physical analysis tool allows a user to show and interpret the physical results of athletic movements, which would be very helpful in preventing long-term injuries. Furthermore, it allows trainers to improve athletic techniques and performance by comparing different athletic motions with each other. A simplified realtime version of our system could also be used effectively in interactive training, offering audio and visual biofeedback to the players during athletic gesture.

A. Annexe

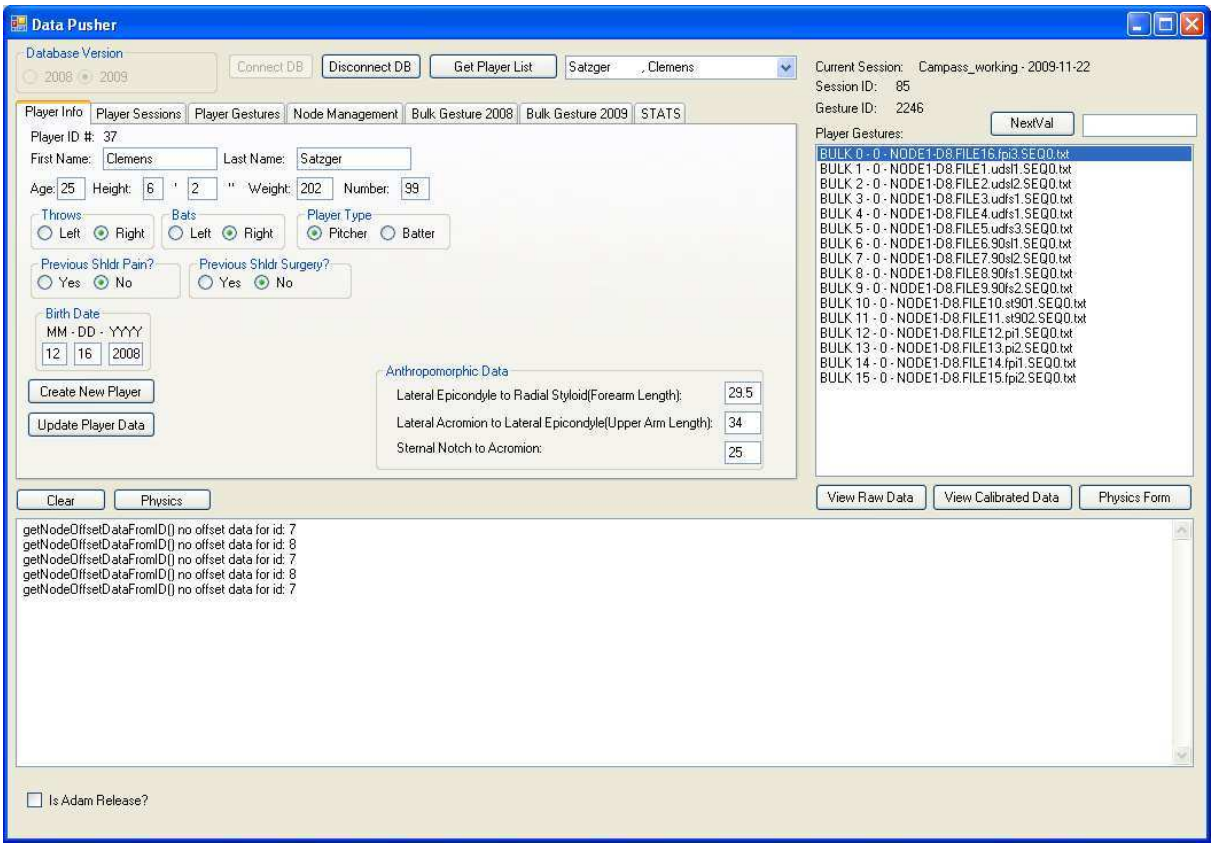

Figure A.1: User Interface Main Window

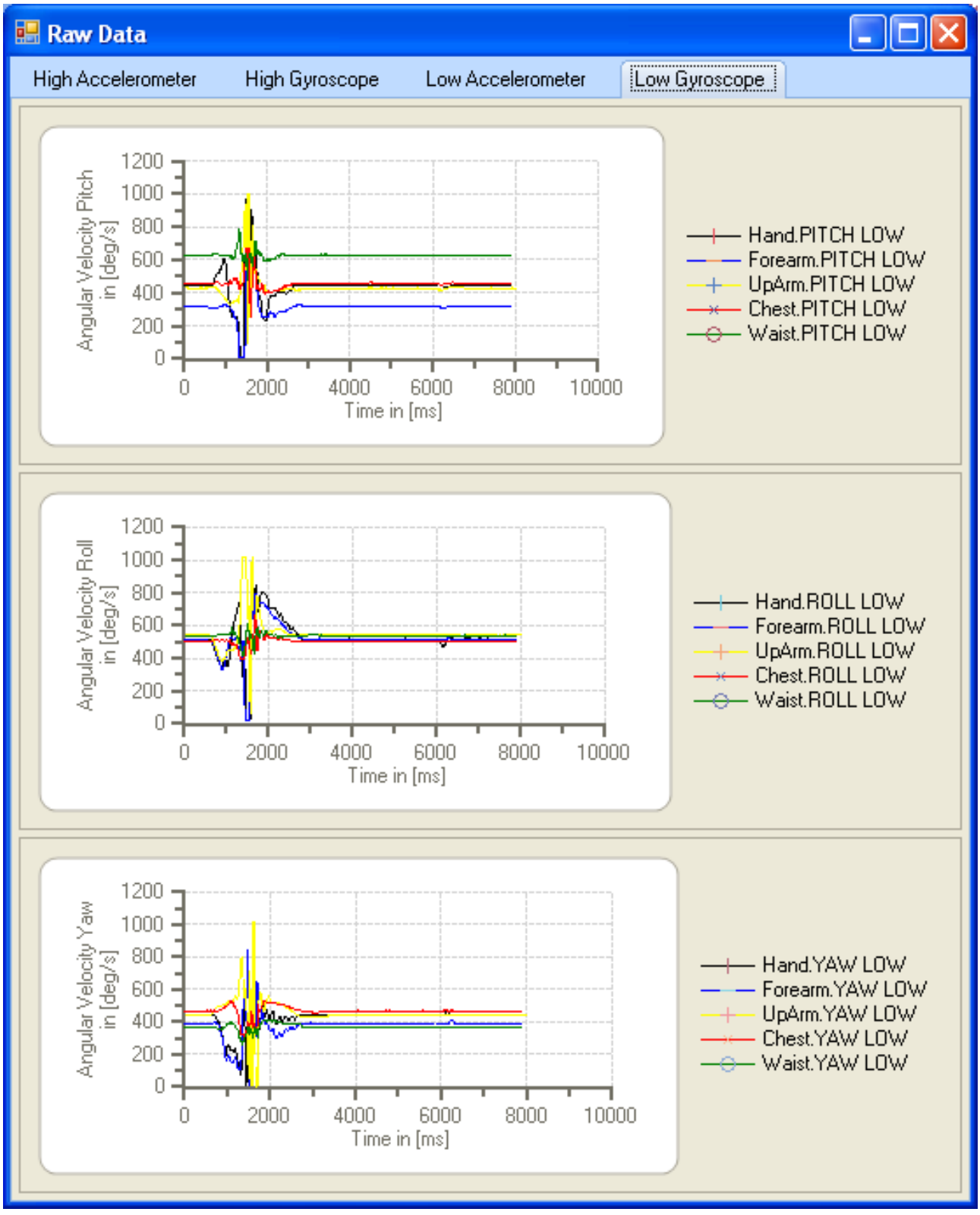

Figure A.2: User Interface Raw Data Window

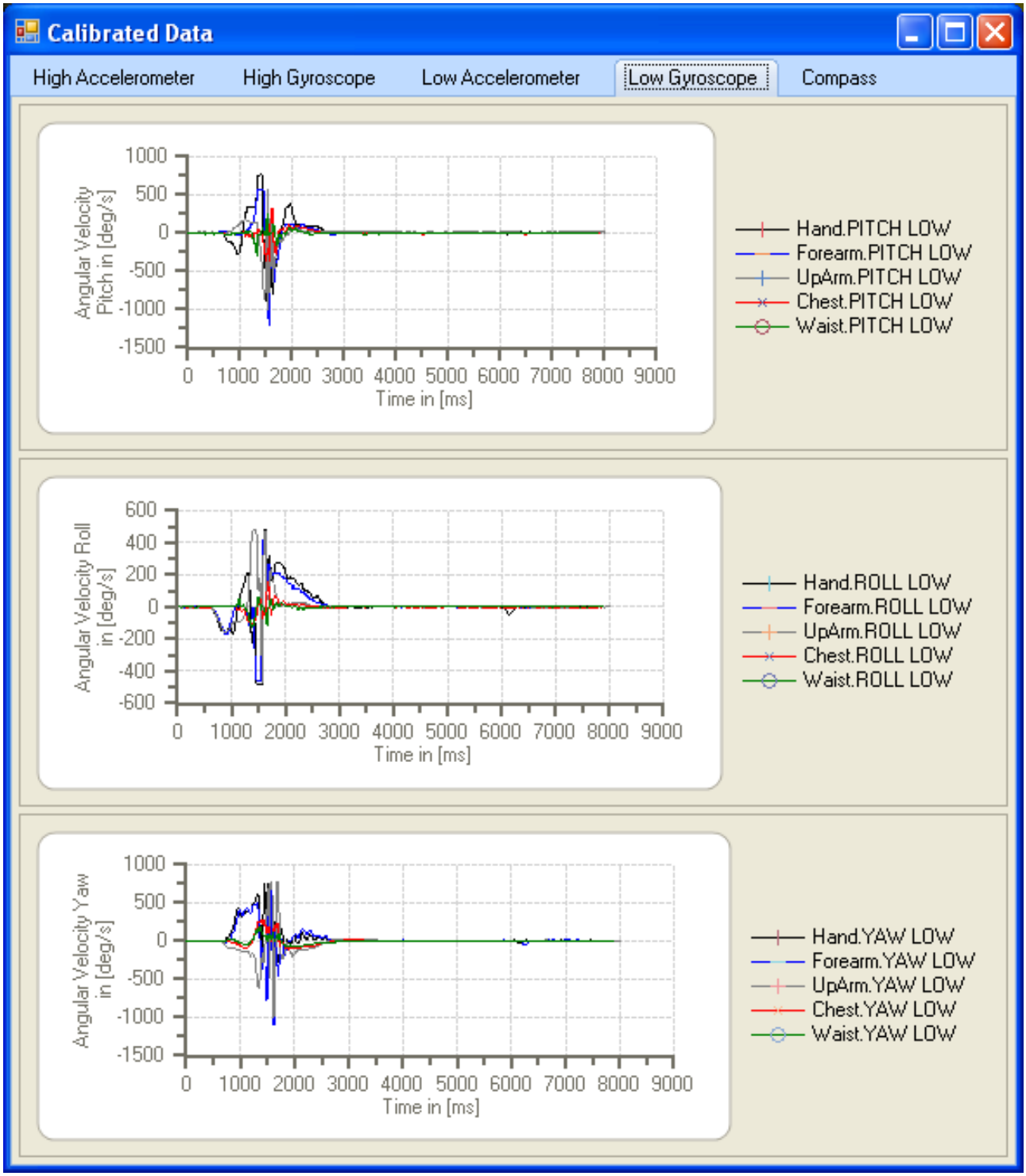

Figure A.3: User Interface Calibrated Data Window

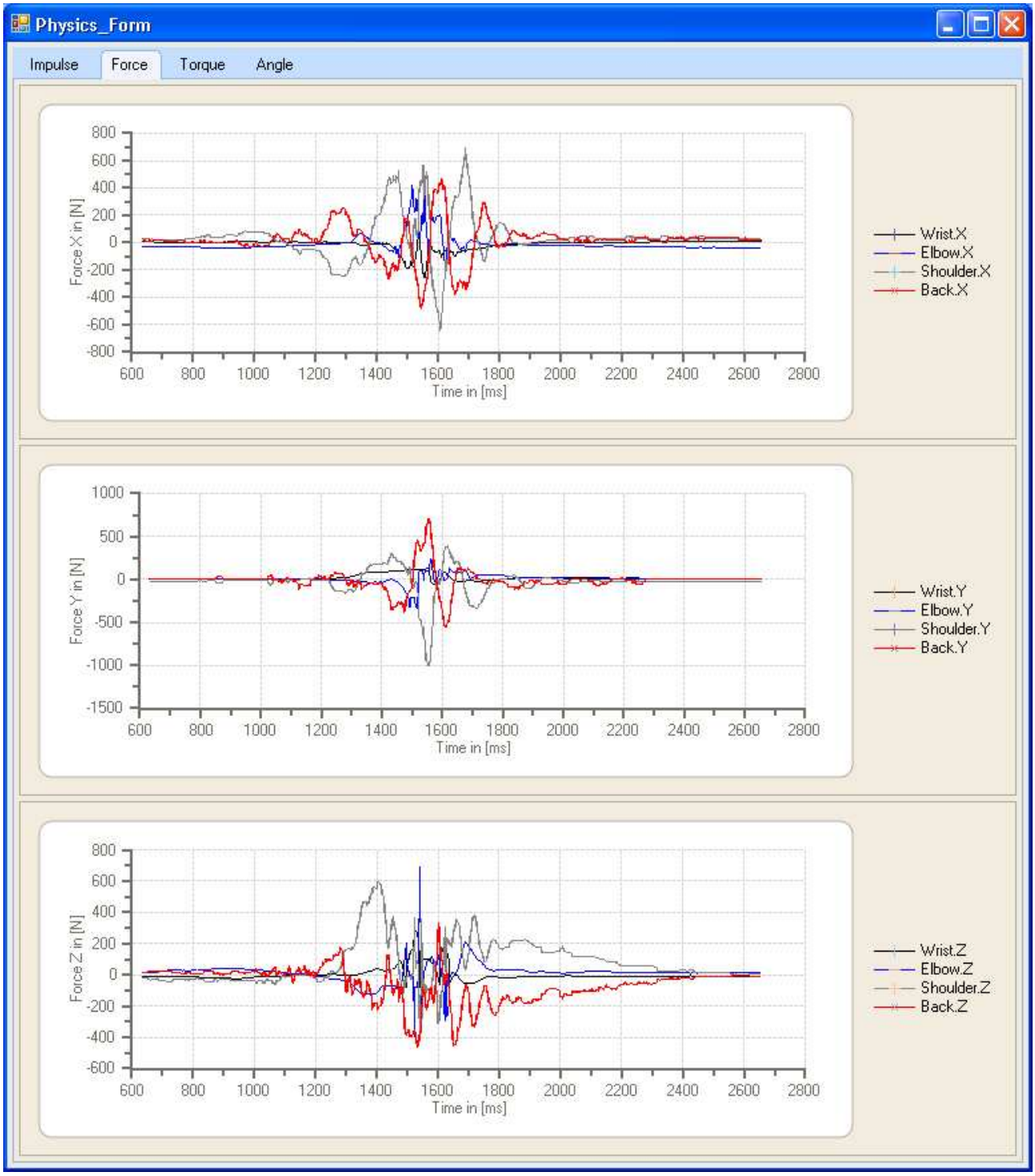

Figure A.4: User Interface Physics Form Window

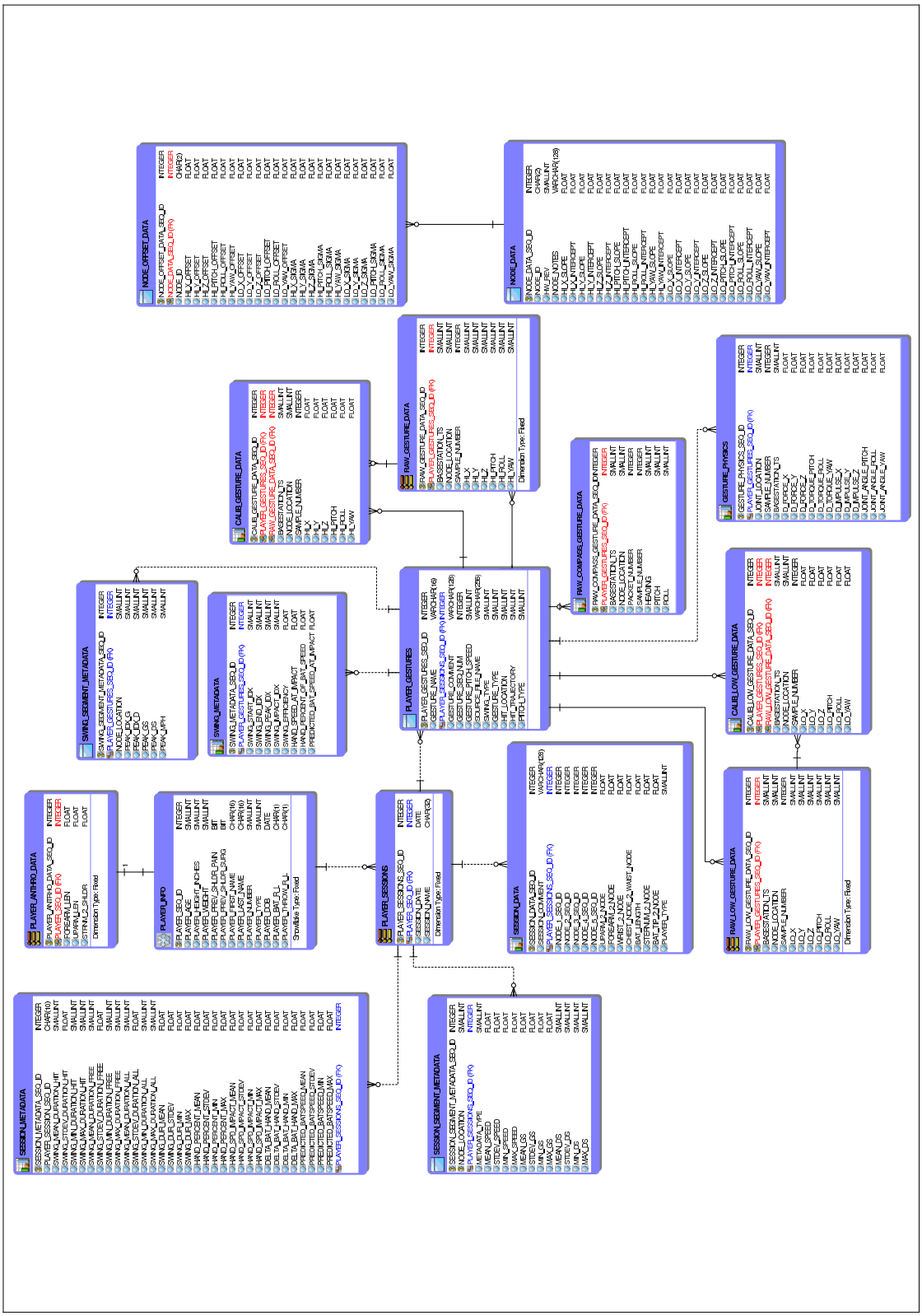

Figure A.5: Database Structure

```
#pragma once
#include <iostream>
#include "havok.h"
#include "newmatap.h"                    // need matrix applications
#include "newmatio.h"                    // need matrix output routines
//using namespace std;

class Kalman
{
  float varG,varQ;
  float timestep;
  IdentityMatrix *I3,*I4;
  Matrix *Xi;
  Matrix *Sigma;
  Matrix *Akmin1;
  Matrix *Kk;
  Matrix *Pkmin,*Pk,*Pkmin1;
  Matrix *Qkmin1;
  Matrix *Rk;
  ColumnVector *qk,*qkmin1,*qkmin,*qmeasure;
public:
  Kalman(float Pk0,float varG,float varQ,float tstep);
  ~Kalman();
  hkQuaternion Kalman::ApplyKalman(hkVector4& w,hkQuaternion& qmeas,hkQuaternion& qk_in);
};
```

Listing A.1: Implementation of the Kalman Filter Header File: Kalman.h

```
#include "Kalman.h"


Kalman::Kalman(float Pk0,float G,float Q,float tstep)
{
  printf("Constructor Kalman\n");
  timestep=tstep;
  varG=G;
  varQ=Q;
  Xi=new Matrix(4,3);
  Sigma=new Matrix(4,4);
  I3=new IdentityMatrix(3);
  I4=new IdentityMatrix(4);
  Akmin1=new Matrix(4,4);
  Kk=new Matrix(4,4);
  Pkmin=new Matrix(4,4);
  Pk=new Matrix(4,4);
  Pkmin1=new Matrix(4,4);
  Qkmin1=new Matrix(4,4);
  Rk=new Matrix(4,4);
  qk=new ColumnVector(4);
  qkmin1=new ColumnVector(4);
  qkmin=new ColumnVector(4);
  qmeasure=new ColumnVector(4);
  *Pk=(*I4)*Pk0;
  *qk=0.0;
}

Kalman::~Kalman()
{
  printf("Destructor Kalman\n");
  delete Xi;
  delete Sigma;
  delete I3;
  delete I4;
  delete Akmin1;
  delete Kk;
```

```
    delete Pkmin;
39  delete Pk;
    delete Pkmin1;
41  delete Qkmin1;
    delete Rk;
43  delete qk;
    delete qkmin1;
45  delete qkmin;
    delete qmeasure;
47 }

49 hkQuaternion Kalman::ApplyKalman(hkVector4& w,hkQuaternion& qmeas,hkQuaternion& qk_in)
   {
51  Real qks;
    *Pkmin1=*Pk;  //next timestep
53  //cout<<"Pkmin1 ="<<endl<<*Pkmin1;
    (*qkmin1)(1) = qk_in(0);
55  (*qkmin1)(2) = qk_in(1);
    (*qkmin1)(3) = qk_in(2);
57  (*qkmin1)(4) = qk_in(3);

59  //cout<<"qk ="<<endl<<*qk;
    //cout<<"qkmin1 ="<<endl<<*qkmin1;
61  *Rk=varQ*(*I4);
    //cout<<"Rk ="<<endl<<*Rk;
63  *Xi<<(*qkmin1)(4)<<-(*qkmin1)(3)<<(*qkmin1)(2)
        <<(*qkmin1)(3)<<(*qkmin1)(4)<<-(*qkmin1)(1)
65      <<-(*qkmin1)(2)<<(*qkmin1)(1)<<(*qkmin1)(4)
      <<-(*qkmin1)(1)<<-(*qkmin1)(2)<<-(*qkmin1)(3);
67  //cout<<"Xi ="<<endl<<*Xi;
    *Sigma=varG*(*I3);
69  //cout<<"Sigma ="<<endl<<*Sigma;
    (*qmeasure)(1)=qmeas(0);
71  (*qmeasure)(2)=qmeas(1);
    (*qmeasure)(3)=qmeas(2);
73  (*qmeasure)(4)=qmeas(3);
    //cout<<"qmeasure ="<<endl<<*qmeasure;
75  Akmin1->Row(1)<<1.0<<timestep*w(2)/2.0<<-timestep*w(1)/2.0<<timestep*w(0)/2.0;
    Akmin1->Row(2)<<-timestep*w(2)/2.0<<1.0<<timestep*w(0)/2.0<<timestep*w(1)/2.0;
77  Akmin1->Row(3)<<timestep*w(1)/2.0<<-timestep*w(0)/2.0<<1.0<<timestep*w(2)/2.0;
    Akmin1->Row(4)<<-timestep*w(0)/2.0<<-timestep*w(1)/2.0<<-timestep*w(2)/2.0<<1.0;
79  //cout<<"A ="<<endl<<*Akmin1;
    //cout<<"At ="<<endl<<Akmin1->t();
81  *Qkmin1=(timestep/2.0)*(timestep/2.0)*(*Xi)*(*Sigma)*(Xi->t());
    //cout<<"Qkmin1 ="<<endl<<*Qkmin1;
83
    ////////Time Update////////////////////
85  *qkmin=(*Akmin1)*(*qkmin1);
    //cout<<"qkmin ="<<endl<<*qkmin;
87  *Pkmin=(*Akmin1)*(*Pkmin1)*(Akmin1->t())+(*Qkmin1);
    //cout<<"Pkmin ="<<endl<<*Pkmin;
89
    ////normalization of qkmin
91
qks=sqrt((*qkmin)(1)*(*qkmin)(1)+(*qkmin)(2)*(*qkmin)(2)+(*qkmin)(3)*(*qkmin)(3)+(*qkmin)
    (4)*(*qkmin)(4));
93  (*qkmin)(1)=(*qkmin)(1)/qks;
    (*qkmin)(2)=(*qkmin)(2)/qks;
95  (*qkmin)(3)=(*qkmin)(3)/qks;
    (*qkmin)(4)=(*qkmin)(4)/qks;
97


99


101
```

```
103     ////////Measurement Update//////////////
        Matrix* PkminRk=new Matrix(4,4);
105     *PkminRk=(*Pkmin)+(*Rk);
        //cout<<"PkminRk ="<<endl<<*PkminRk;
107     *Kk=(*Pkmin)*((PkminRk)->i());
        //cout<<"PkminRk-1 ="<<endl<<(PkminRk->i());
109     //cout<<"Kk ="<<endl<<*Kk;
        *qk=(*qkmin)+(*Kk)*((*qmeasure)-(*qkmin));
111     //cout<<"qk ="<<endl<<*qk;
        *Pk=((*I4)-(*Kk))*(*Pkmin);
113     //cout<<"Pk ="<<endl<<*Pk;

115     ////normalization of qk
        qks=sqrt((*qk)(1)*(*qk)(1)+(*qk)(2)*(*qk)(2)+(*qk)(3)*(*qk)(3)+(*qk)(4)*(*qk)(4));
117     (*qk)(1)=(*qk)(1)/qks;
        (*qk)(2)=(*qk)(2)/qks;
119     (*qk)(3)=(*qk)(3)/qks;
        (*qk)(4)=(*qk)(4)/qks;
121     //cout<<"qk normalized ="<<endl<<*qk;
        hkQuaternion q((*qk)(1),(*qk)(2),(*qk)(3),(*qk)(4));
123     //q.normalize();
        delete PkminRk;
125     return q;
    }
```

Listing A.2: Implementation of the Kalman Filter Header File: Kalman.cpp

B. Glossary

- **Accelerometer**: System that measures linear accelerations.
- **Gyroscope**: System that measures angular velocities.
- **Magnetometer**: System that measures magnetic fields.
- **Digital Compass**: System out of an accelerometer and a magnetometer to measure an orientation in 3 dimensions.
- **Ragdoll**: A ragdoll is a collection of multiple rigid bodies (each of which is ordinarily tied to a bone in the graphics engine's skeletal animation system) tied together by a system of constraints that restrict how the bones may move relative to each other.
- **IMU**: Inertial measurement unit. It is a system of gyroscopes and accelerometers.
- **IMS**: Inertial measurement system. It is a system out of inertial measurement units.
- **Database**: A Database is an integrated collection of logically related records or files consolidated into a common pool that provides data for one or more multiple uses.
- **dll**: Dynamic-link library. Dynamic-link library (also written without the hyphen), or DLL, is Microsoft's implementation of the shared library concept in the Microsoft Windows and OS/2 operating systems.
- **anthropometric**: The science of measuring the human body as to height, weight, and size of component parts, including skinfold thickness, to study and compare the relative proportions under normal and abnormal conditions.
- **GUI**: A graphical user interface is a type of user interface item that allows people to interact with programs in more ways than typing such as computers.
- **Low-G**: Low accelerations below $\pm 6\,\mathrm{g}$
- **High-G**: High accelerations higher than $\pm 6\,\mathrm{g}$
- **I2C**: Inter-Integrated Circuit is a multi-master serial computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, or cellphone.
- **SPI**: The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard named by Motorola that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame.
- **(Gaussian) Standard deviation**: In probability theory and statistics, the standard deviation of a statistical population, a data set, or a probability distribution is the square root of its variance. It shows how much variation there is from the "average" (mean).
- **Sigma**: That is, the standard deviation sigma is the square root of the average value of $\sigma = \sqrt{\mathrm{E}\left[(X-\mu)^2\right]}$.

- **Ksdensity**: In statistics, kernel density estimation is a non-parametric way of estimating the probability density function of a random variable.
- **Euler Angle**: The Euler angles were developed by Leonhard Euler to describe the orientation of a rigid body in 3-dimensional Euclidean space.

- **Quaternion**: In mathematics, the quaternions are a number system that extends the complex numbers. They were first described by Irish mathematician Sir William Rowan Hamilton in 1843 and applied to mechanics in three-dimensional space.
- **Covariance**: In probability theory and statistics, covariance is a measure of how much two variables change together.
- **Impingement**: Impingement syndrome, also called painful arc syndrome, supraspinatus syndrome, swimmer's shoulder, and thrower's shoulder, is a clinical syndrome which occurs when the tendons of the rotator cuff muscles become irritated and inflamed as they pass through the subacromial space, the passage beneath the acromion. This can result in pain, weakness and loss of movement at the shoulder.
- **multibody**: A multibody system is used to model the dynamic behavior of interconnected rigid or flexible bodies, each of which may undergo large translational and rotational displacements.

C. Symbols and Abbreviations

a_k	Acceleration vector in x,y,z
v_k	Velocity vector in x,y,z
p_k	Position vector in x,y,z
ω	Angular velocity vector in pitch, roll, yaw
φ	Angle vector in pitch, roll, yaw
F	Force
m	Mass
Δv	Difference of velocity
Δt	Difference of time
ΔI	Difference of Impulse
I_{t-1}, I_t	Impulse at one time step before and at this time step
time step	Time step of the simulation
T	Torque
e_i	Change of position
t_i	Target position
s_i	End effector position
θ	Joint angle
J	Matrix of Inertia from the center of the joint
$\Delta\vec{s}$	Vector of end effectors
$J(\theta)$	Jacobian matrix
$\Delta\theta$	Change of the joint angle
p_j	The jth position of the joint
v_j	The jth unit vector along the current axis of rotation
θ_j	The jth joint angle
s_i	The ith end effector position
$\dot{\omega}$	Angular velocity derived by the time
ω_{t-1}, ω_t	Angular velocity at time $t-1$ ant at time t
J_{ij}	Entries of the matrix of inertia around the joint
I_{ij}	Entries of the matrix of inertia around the center of gravity
$\vec{a}$	Displacement vector $(a_1x + a_2y + a_3z)$ from the center of gravity to the new axis
δ_{ij}	Kronecker delta
$\hat{X}_k$	Current Estimation
$\hat{X}_{k-1}$	Previous estimation
K_k	Kalman gain
Z_k	Measured Value
x_k, x_{k-1}	Actual and previous state
z_k	Observation
u_k	Control inputs

w_{k-1}	Process noise
v_k	Measurement noise
A	State time evolution
B	Control to state space conversion
H	Input to state space conversion(measurement sensibility matrix)
$\hat{x}$	Estimation of x
$\hat{x}^-$ or $\hat{x}^+$	A priori or a posteriori estimation of x
w_{k-1}	Process noise
Q_{k-1}	Process noise covariance
R_k	Measurement noise covariance
K_k	Kalman matrix
P_k^-	A priori error covariance
P_k	Error covariance
$\dot{f}(t)$	Derivation of the function $f(t)$ over time
$f(t + \Delta t)$	The function $f(t)$ one time step later
$f(t)$	The function f depending on the time
Δt	Time step
q_{k-1}	Last quaternion operator
$\Omega(\omega)$	Angular velocity matrix
1	Identity matrix
w_{k-1}	Process noise
Δt	Time step
Ξ_{k-1}	Transformation matrix
$n_{G,k-1}$	Measurement noise
y_G	Sensor signal gyroscope
Q_{k-1}	Noise covariance of the gyroscope
$\sum_G$	Covariance matrix of the gyroscope measurement noise

List of Figures

List of Tables

Bibliography

[AB08] D. K. Arvind and Andrew Bates. The speckled golfer. In *BodyNets '08: Proceedings of the ICST 3rd international conference on Body area networks*, pages 1–7, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 8

[A.D06] A.D. Young, M.J. Ling, D.K. Arvind. Orient-2: A realtime wireless posture tracking system using local orientation estimation. In *4th Workshop on Embedded Networked Sensors: EmNets 2007*, pages 53–57, November 2006. 6

[Asc09] Ascension Technology Corporation. http://www.ascension-tech.com/, 09/15/2009. 6

[Atm] Atmel. Idg-300 integrated dual-axis gyroscope. http://www.atmel.com/products/avr32/default.asp. 15

[Ayl06] Ryan Aylward. Sensemble: A wireless inertial sensor system for interactive dance and collective motion analysis. Master's thesis, Media Laboratory, Massachusetts Institute of Technology, 2006. 5, 7, 9

[Bam07] Bamberg, S.J.M., Benbasat A.Y., Scarborough D.M., Krebs D.E., Paradiso J.A. Wearable wireless sensor network to assess clinical status in patients with neurological disorders. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 563–564, 2007. 5

[BDB06] MD Brian D. Busconi. Instability of the athlete's shoulder. April 7 2006. 4

[Ben00] Ari Y. Benbasat. An Inertial Measurement Unit for User Interfaces. Master's thesis, Media Laboratory, Massachusetts Institute of Technology, 2000. 6

[Ben07] Benbasat, A.Y. and Paradiso, J.A. Groggy wakeup - automated generation of power-efficient detection hierarchies for wearable sensors. In *Proceedings of the 4th International Workshop on Wearable and Implantable Body Sensor Networks (BSN 2007)*, pages 59–64, March 2007. 6

[Ber07] Berkson E., Aylward R., Zachazewski J., Paradiso J., Gill T.J. Imu arrays: The biomechanics of baseball pitching. In *The Orthopaedic Journal at Harvard Medical School, Vol. 8*, pages 90–94, June 2007. 1, 5

[Bög06] Alfred Böge. *Technische Mechanik*. Vieweg Fachbücher der Technik, 27th

edition edition, 2006. 47

[Bio09] Biometrics Ltd UK, 08/12/2009. http://www.biometricsltd.com/. 7

[Bud08] Buddhika de Silva, Anirudh Natarajan, Mehul Motani, Kee-Chiang Chua. A real-time feedback utility with body sensor networks. In *5th International Workshop on Wearable and Implantable Body Sensor Networks*, pages 49–53, June 2008. 6

[Bus04] Samuel R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. April 2004. 37

[cCTRLtH09] c Copyright Telekinesys Research Ltd. (t/a Havok). Havok physics animation user guide. 1999-2009. 25, 36, 64

[CK00] Eck JC Covington LA Chambless KM, Knudtson J. Rate of injury in minor league baseball by level of play. *Am J Orthop*, 2000. 2

[CS01] Garrick JG Conte S, Requa RK. Disability days in major league baseball. *Am J Sports Med*, 2001. 1

[D. 03] D. Fontaine, D. David, Y. Caritu. Sourceless human body motion capture. In *Smart Objects Conference*, May 2003. 6

[Dan07] Daniel Vlasic, Jovan Popovic. Mit computer science and artificial intelligence labratory. In *CSAIL Research Abstracts 2007*, page http://publications.csail.mit.edu/abstracts/abstracts07/drdaniel_2007/drdaniel.html, 2007. 6

[DRPHV07] Per J. Slycke Daniel Roetenberg and Member IEEE Peter H. Veltnik. Ambulatory position and oriantation tracking fusing magnetic and inertial sensing. May 2007. 7

[ERB01] Xiaoping Yun Michael J. Zyda Eric R. Bachmann, Robert B. McGhee. Inertial and magnetic posture tracking for inserting humans into networked virtual environments. *VRST'01*, November 15-17 2001. 7

[FG95] Dillman CJ Escamilla RF Fleisig GS, Andrews JR. Kinetics of baseball pitching with implications about injury mechanisms. *Am J Sports Med*, 1995. 2

[FG96] Andrews JR. Fleisig GS, Escamilla RF. Biomechanics of throwing. 1996. 2, 3, 68, 72, 74, 96

[Gar06] Helmut Garstenauer. A unified framework for rigid body dynamics. Master's thesis, Johannes Kepler Universitaet Linz, 2006. 35

[GW05] H.E.J. (DirkJan) Veeger Mohsen Makhsous Peter Van Roy Carolyn Anglin Jochem Nagels Andrew R. Karduna Kevin McQuade Xuguang Wang Frederick W. Werner Bryan Buchholz Ge Wua, Frans C.T. van der Helm. Isb recommendation on definitions of joint coordinate systems of various joints for the reporting of human joint motion part ii: shoulder, elbow, wrist and hand. *Journal of Biomechanics*, 2005. 42

[Har09] Harvey Weinberg. AN-625: Modifying the Range of the ADXRS150 and ADXRS300 Rate Gyro, 09/10/2009. http://www.analog.com/static/imported-

files/application_notes/591355220429067903767268819AN625_0.pdf. 12

[Hon09] Honeywell. 3-Axis Digital Compass Module HMC6343, 09/10/2009. http://www.ssec.honeywell.com/magnetic/datasheets/HMC6343.pdf. 12

[htt09] http://opensimulator.org/wiki/Development_Team. http://www.avatarmovie.com/index.html, 01/21/2009. 36

[Inc09] Analog Devices Inc. www.analog.com/en/mems-and-sensors/imems-gyroscopes/adxrs300/products/product.html, Aug 09/10/2009. 11, 17

[Inc10] Analog Devices Inc. Adxl193: Single-axis, high-g, imems® accelerometers, 01/23/2010. http://www.analog.com/en/mems-and-sensors/imems-accelerometers/ADXL193/products/product.html. 12, 18

[Inv] InvenSense. AVR32 32-bit RISC processor Microcontroller. http://www.invensense.com/products/idg_300.html. 11

[JAC08] F. Torres J. A. Corrales, F. A. Candelas. Hybrid tracking of human operators using imu/uwb data fusion by a kalman filter. In *ACM/IEEE International Conference on Human-Robot Interaction*, 2008. 8

[JCKC92] IEEE Jack C. K. Chou, Member. Quaternion kinematic and dynamic differential equations. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, Vol 8, 1992. 55, 56

[J.F06] D.Kumar D.Fitzgerald T.Ward B.Caulfield C.Markham J.Foody, D.Kelly. A prototype sourceless kinematic-feedback based video game for movement based exercise. 2006. 7

[JP00] A. Benbasat Z. Teegarden J. Paradiso, K. Hsiao. Design and implementation of expressive footwear. *IBM Systems Journal*, pages 511–529, Volume 39, October 2000. 6

[JTDJ09] Derrick Fluhme Karen J. Mohr Lewis A. Yocum Neal S. ElAttrache J. T. Davis, Orr Limpisvasti and Frank W. Jobe. The effect of pitching biomechanics on the upper extremity in youth and adolescent baseball pitchers. *The American Journal of Sports Medicine*, 2009. 2, 4

[Koo04] Koon Kiat Teu, Wangdo Kim, Franz Konstantin Fuss. Using dual number method for motion analysis of left arm in a golf swing. In *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, pages 217–220, June 2004. 7

[Kwa08] Kwang Yon Lim, Wei Dong, Francis Young Koon Goh, Kim Doang Nguyen, I-Ming Chen, Song Huat Yeo, Henry Been Lirn Duh. A preliminary study on the accuracy of wireless sensor fusion for biomotion capture. In *5th International Workshop on Wearable and Implantable Body Sensor Networks*, pages 99–102, June 2008. 6

[LA07] Guang-Zhong Yang Omer Aziz Louis Atallah, Benny Lo. Detecting walking gait impairment with an ear-worn sensor. 2007. 5

[Lam] T. Y. Lam. Hamilton's quaternions. 55

[Lap08] Michael Lapinski. A wearable, wireless sensor system for sports medicine. Master's thesis, Massachusetts Institute of Technology, September 2008. 1, 7, 12, 18, 33

[LEJP09] Jung Keun Lee and IEEE Edward J. Park, Member. A minimum-order kalman filter for ambulatory real-time human body orientation tracking. 2009 IEEE International Conference on Robotics and Automation, May 2009. 7, 44

[LM67] J. N. Lyness and C. B. Moler. *Numerical differentiation of analytic functions*. Society for Industrial and Applied Mathematics (SIAM), 4th edition edition, 1967. 3

[Lor07] Lorincz K., Kuris B., Ayer S.M., Patel S., Bonato P., Welsh M. Wearable wireless sensor network to assess clinical status in patients with neurological disorders, cambridge ma. In *International Conference on Information Processing in Sensor Networks*, pages 413–423, April 25-27, 2007. 6

[MAT10] MATLAB. Kernel smoothing density estimate, 01/10/2010. http://www.mathworks.com/access/helpdesk/help/toolbox/stats/index.html?/access/helpdesk/help/toolbox/stats/ksdensity.html&http://www.google.com/search?q=ksdensity+matlab&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official&client=firefox-a. 22

[MB06] M.D. Michael Brown. Superior labrum, anterior posterior (slap) lesions. April 7 2006. 4

[Mea09] Measurand Inc. http://www.motion-capture-system.com/, 09/10/2009. 6

[Min09] MiniSun LLC . Intelligent device for energy expenditure and physical activity, 10/12/2009. http://www.minisun.com/. 6

[Mot09] Motion Analysis Inc. Motion analysis, 11/10/2009. http://www.motionanalysis.com/. 4

[MS01] Angus P. Andrews Mohinder S.Grewal. *Kalman Filtering Theory and Practice Using MATLAB*. Wiley, second edition edition, 2001. 8, 52

[Mya09] Myapex. bodybugg, 12/10/2009. http://my.apexfitness.com/vip/bb_enrollment_info_public.php. 5, 81

[NAD06] M.D. Nicola A. DeAngelis. Impingement. April 7 2006. 4

[Nik09] Nike. http://nikerunning.nike.com/nikeos/p/nikeplus/en_us/, 01/21/2009. 5, 81

[Nor09] Nordic Semiconductor. nRF2401A Transceiver , 09/10/2009. http://www.nordicsemi.com/index.cfm?obj=product&act=display&pro=64#. 13

[OS84] D. E. Orin and W. W. Schrader. Efficient computation of the jacobian for robot manipulators. *International Journal of Robotics Research*, pages pp. 66–75., 1984. 38

[Pol] Polhemus. http://www.polhemus.com/. 6

[Ric73] Richard T. Weidner and Robert L. Sells. *Elementary Classical Physics,*

Second Edition, volume 1. Allyn and Bacon, 1973. 19

[RL01] Uribe JW Rizio L. Overuse injuries of the upper extremity in baseball. *Clin Sports Med*, 2001. 2

[Sch09] Michael S. Schmidt. Baseball injuries rise, and reason is a mystery. *New York Times*, July 7 2009. 4

[tCF09] 20th Century Fox. http://opensimulator.org/wiki/main_page, 01/21/2009. 35

[TH73] King J. Tullos HS. Throwing mechanism in sports. *Orthop Clin North*, 1973. 1, 2

[TJ72] Sweterlitsch P Torg JS, Pollack H. The effect of competitive pitching on the shoulders and elbows of preadolescent baseball players. *Pediatrics*, 1972. 2

[T.W80] T.W. Calvert, Jo Chapman, A. Patla. The integration of subjective and objective data in the animation of human movement. In *ACM SIGGRAPH Computer Graphics, Volume 14, Issue 3*, pages 198–203, 1980. 7

[W. 08] W. Yeoh, J. Wu, I. Pek, Y. Yong, X. Chen, A.B. Waluyo . Real-time tracking of flexion angle by using wearable accelerometer sensors. In *5th International Workshop on Wearable and Implantable Body Sensor Networks*, pages 125–128, June 2008. 8

[WB06] Greg Welch and Gary Bishop. An introduction to the kalman filter. 07/24/2006. 54

[Wik10] Wiktionary. data fusion, 01/23/2010. http://en.wiktionary.org/wiki/data_fusion. 35

[Win05] David A. Winter. *Biomechanics and Motor Control of Human Movement.* Johne Willey & Sons, third edition edition, 2005. 27, 44

[XOS09] XOS Technologies. Xos technologies, 08/07/2009. http://www.xostech.com/. 4

[Xse09] Xsens Technologies B.V. Xsens motion technologies, 11/11/2009. http://www.xsens.com. 7, 8

[XYERB06] IEEE Xiaoping Yun, Fellow and IEEE Eric R. Bachmann, Member. Design, implementation, and experimental results of a quaternion-based kalman filter for human body motion tracking. *IEEE Transactions on Robotics*, Vol. 22, 2006. 53

[Yan06] Guang-Zhong Yang. *Body Sensor Networks.* Springer, 2006. 13

[You09] A.D. Young. Comparison of orientation filter algorithms for realtime wireless inertial posture tracking. *Body Sensor Networks*, 2009. 7, 11

[ZWS09] Andreas Zinnen, Christian Wojek, and Bernt Schiele. Multi activity recognition based on bodymodel-derived primitives. In *LoCA*, pages 1–18, 2009. 8