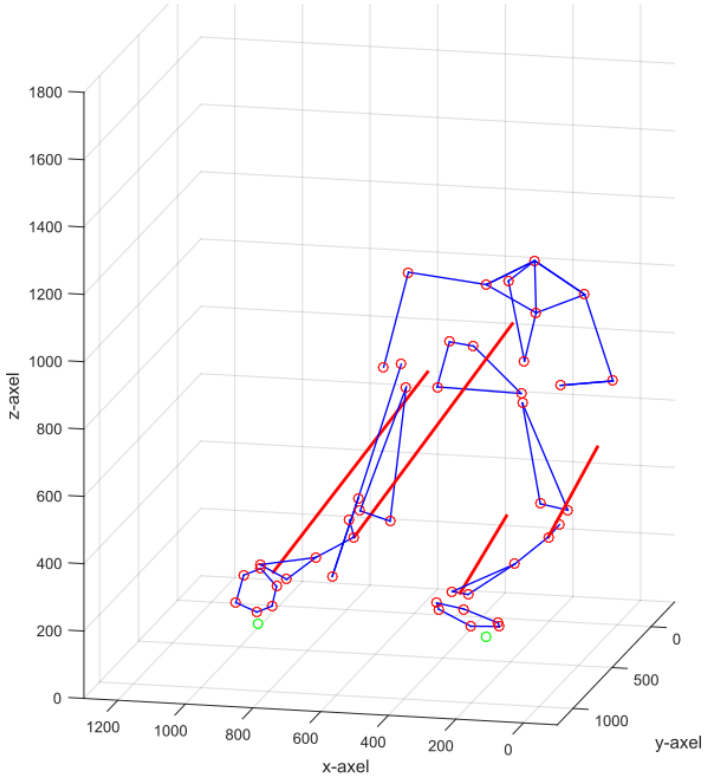
CHALMERS



Analys av rörelsemönster för kvinnliga innebandyspelare

Framtagning av en metod för kvantitativa undersökningar

Kandidatarbete inom civilingenjörsprogrammen Maskinteknik och Teknisk fysik

Simon Hannonsson
Alexander Olsson
Lovisa Westlund Gotby

Kanditatarbete TIFX02-15-11
Institutionen för teknisk fysik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2015

Analys av rörelsemönster för kvinnliga innebandyspelare
Framtagning av en metod för kvantitativa undersökningar
SIMON HANNONSSON
ALEXANDER OLSSON
LOVISA WESTLUND GOTBY
Institutionen för teknisk fysik
Chalmers tekniska högskola

Sammandrag

Detta kandidatarbete har haft som mål att, genom biomekanisk modellering, ta fram en metod som tillåter analys av kvinnliga innebandyspelares rörelsemönster under en rörelsesekvens karaktäristisk för sporten. Sekvensen som har analyseras är en kort löpsekvens med en vändning på 180 grader. Data som beskriver rörelsen och markreaktionskrafter i samband med vändningen har samlats in med hjälp av kamerasystemet Qualisys Motion Capture med tillhörande kraftplattor. Denna data har sedan behandlats i progamvaran Qualisys Track Manager för att efter det kunna exporteras till beräkningsprogrammet MATLAB.

Genom ingejörsmässiga beräkningar och antaganden har sedan en metod tagits fram som kan identifiera kvantitativa skillnader mellan olika spelare och därmed ge indikationer på vad en effektiv rörelse är. Detta har gjorts genom ett program som är kodat i MATLAB. Programmet läser in insamlad data och beräknar utifrån denna relevanta vinklar, hastigheter, accelerationer och krafter som representerar kinematiken och den inversa dynamiken för rörelsen.

Mätningar har gjorts på tre olika spelare då de utför den valda rörelsesekvensen. Alla viktiga resultat som beräknats från mätdatan presenteras numeriskt i tabell samt visualiseras i grafer; de olika spelarnas vändningar är således kvantitativt jämförbara.

Nyckelord: innebandy, Qualisys Motion Capture, biomekanisk modellering, kinematik, invers dynamik, rörelsemönster

Abstract

The purpose with this bachelor thesis has been to obtain a method in order to analyse movement pattern for female floorball players by means of biomechanical modelling. The sequence of motion studied in this report is a short running-sequence with a 180 degree turn which is characterising for the game. Data representing the motion and the ground reaction forces in the turning has been collected with the system Qualisys Motion Capture linked with force plates. After this the data was processed by the computer software Qualisys Track Manager and thereafter exported to MATLAB.

A program which execute computations by ways of assumptions validated in biomechanics was then developed in MATLAB. This program imports the collected data and computes the essential angles, velocities, accelerations and forces which describes the kinematics and the inverse dynamics of the motion. Consequently, the method makes it possible to identify quantitative differences for different players and give indications of how to move more efficiently.

Measurements was carried out on three different players when they performed the chosen sequence of motion. All the major numerical results from the computations are presented in a table and the rest are visualised in plots. Thus, the pivot of different players are quantitatively comparable.

Keywords: floorball, Qualisys Motion Capture, biomechanical modelling, kinematics, inverse dynamics, movement pattern

Förord

Denna rapport är resultatet av ett projekt som våren 2015 utfördes inom ramen för kandidatarbeten på Chalmers tekniska högskola. Arbetet syftar till att ta fram en metod för att kvantitativt undersöka rörelsemönster för kvinnliga innebandyspelare. Handledare för projektet var docent Magnus Karlsteen på institutionen för teknisk fysik, Chalmers tekniska högskola, samt doktor Tobias Hein på institutionen för kost- och idrottsvetenskap, Göteborgs universitet. Examinator för arbetet är professor Jan Swenson, institutionen för teknisk fysik på Chalmers tekniska högskola.

Författarnas tack

Kandidatgruppens medlemmar vill börja med att tacka Magnus Karlsteen som handlett gruppen under arbetets gång. Han har varit ett stort stöd, kommit med konstruktiv kritik och gett inspiration. Vi vill också tacka Tobias Hein för att han hjälp oss med den biomekaniska modelleringen och för att vi fått tillgång till mäturustning på Göteborgs universitet samt Frida Bakkman på Qualisys för att vi även där fått göra mätningar.

Vi riktar ett stort tack till Per Tjusberg, Katrinelundsgymnasiets idrottsutbildning, och Marika Greberg, fystränare Pixbo Wallenstam IBK, för att idén till projektet överhuvudtaget föddes och för deras ständiga entusiasm och engagemang under hela våren. Vi vill tillsist också tacka eleverna på Katrinelundsgymnasiet för att de medverkat i det filmmaterial och de mätningar som analyserats.

Simon Hannonsson, Alexander Olsson och Lovisa Westlund Gotby
Göteborg, 2015

Innehåll

1
Introduktion

1.1 Bakgrund

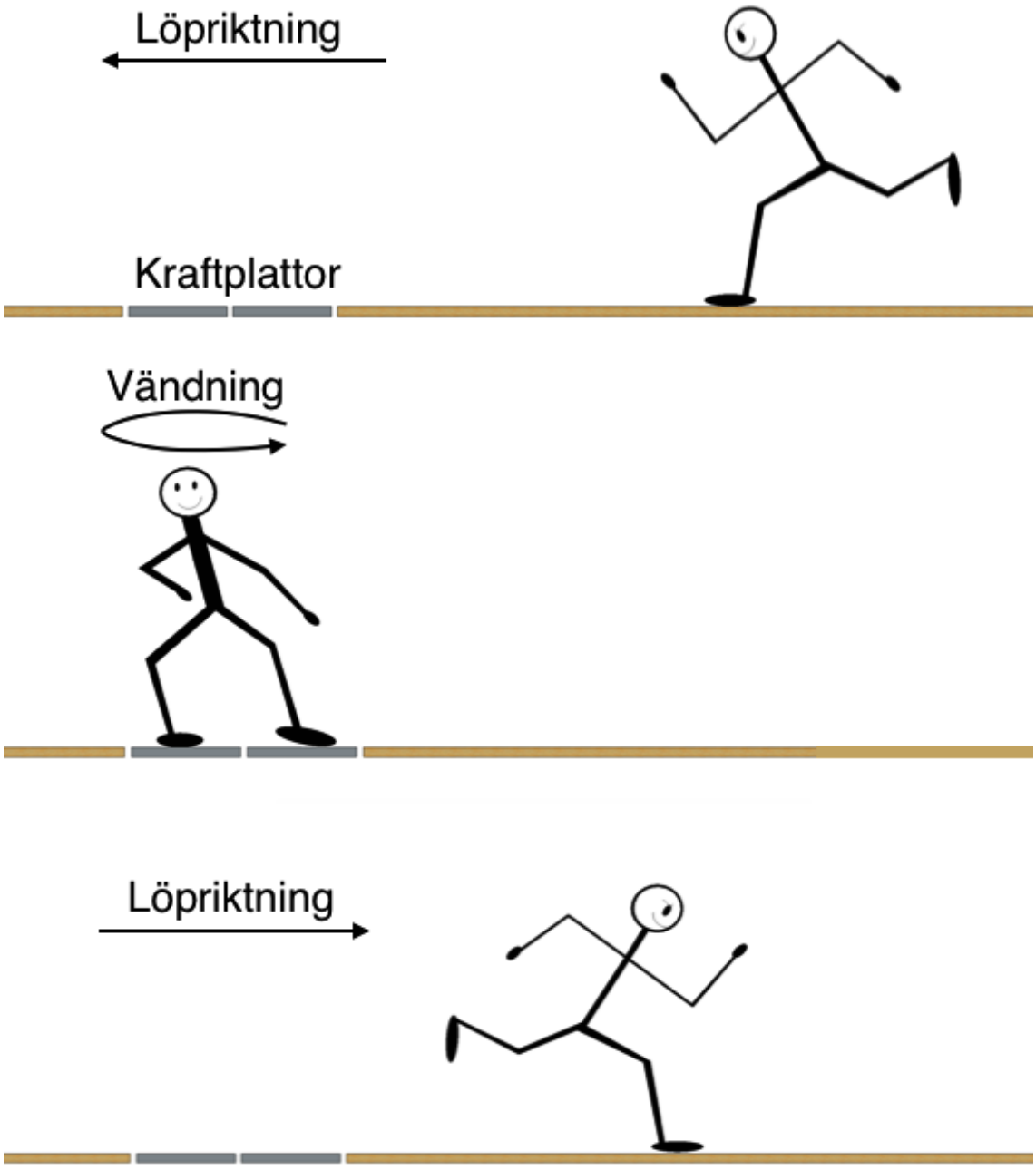
Skador är något som tyvärr är vanligt förekommande i samband med utövande av sport. Dessa sportrelaterade skador är någonting man, genom förebyggande träning, vill förhindra innan de har uppkommit och detta är därför ett område under ständig förändring och förbättring [1]. Innebandy är en sport med många snabba vändningar och därmed är påfrestningarna för spelarna stora på anklar, knän och höfter. Frågeställarna till detta projekt, innebandyklubben Pixbo Wallenstam, vill gärna belysa detta eftersom de tycker att det är viktigt att träna sina spelare på ett långsiktigt hållbart sätt då allt yngre ungdomar idag tränar allt hårdare och förekomsten av skador då blir vanligare.

Tränarna i Pixbo Wallenstam jobbar aktivt med att analysera och att göra sin spelare medvetna om sitt rörelsemönster genom att använda tekniska hjälpmedel som mobilapplikationen Coach's Eye. Denna applikation är ett mycket bra redskap för vardagsbruk eftersom den tillåter användaren att filma en sekvens och spela upp denna i slowmotion samt rita på bilden för att lättare kunna påvisa olika saker [2]. Nackdelen med denna typ av analys är att det förlitar sig helt på användarens egna iakttagelser och erfarenheter och en tolkning kan då bli väldigt subjektiv.

Det här arbetet kommer fokusera på att kvantitativt analysera kvinnliga innebandyspelares rörelsemönster med hjälp av kamerasystemet Qualisys Motion Capture [3]. Dessa kameror samlar in stora mängder data som fullständigt representerar en rörelse och det finns alltså stor potential i att använda detta för att göra en objektiv analys.

1.2 Syfte

Syftet med detta kanditatarbete är att, genom biomekanisk modellering, ta fram en metod som gör att en innebandyspelares rörelsemönster kvantitativt kan analyseras. För att göra detta utförs mätningar på en rörelsesekvens som är representativ för innebandy och data från denna samlas in med hjälp av Qualisys Motion Capture och tillhörande kraftplattor. Kinematiken och den inversa dynamiken för denna rörelse studeras sedan genom beräkningar utförda i MATLAB och resultatet av dessa beräkningar presenteras på ett lättillgängligt sätt. Rörelsen som studeras är en kort löpsekvens som innefattar en vändning på 180 grader illustrerad i figur 1.1.



Figur 1.1: Den korta löpsekvens, med en 180 graders vändning, som studerats i detta arbete. Sekvensen kan delas in i de tre delarna inbromsning, vändning och acceleration. Notera att det rent mätningstekniskt är viktigt att vändningen sker med en fot på vardera kraftplatta.

1.3 Avgränsningar

Detta projekt genomförs inom ramen av ett kandidatarbete och är därmed begränsat i tid till halvtidsarbete under en termin för de tre studenterna i gruppen. Datainsamlingen genomfördes i samarbete med institutionen för kost och idrottsvetenskap vid Göteborgs universitet samt med Qualisys; möjligheten att genomföra mätningar är alltså begränsade. Arbetet kommer därför inte leda till något resultat baserat på statistik utan endast enskilda fall kommer studeras.

En tanke med projektet från början var att det skulle hitta ett optimalt rörelsemönster som både är skademinimerande och så tidseffektivt som möjligt. Inga slutsatser huruvida en rörelse är skadlig eller inte kommer dock dras utan endast indikationer om det kommer ges. Detta eftersom kunskap om varför och om vilka skador som uppkommer i och med olika belastningar är inte en kompetens som gruppen besitter.

2

Teori – Biomekanisk modellering

I följande avsnitt går vi igenom grunderna för den biomekaniska modellering som tillämpats för att betrakta och analysera kinematiken och den inversa dynamiken i detta arbete. All teori baseras på [4] och bygger på data som samlats in med hjälp av kamerasystemet Qualisys Motion Capture samt av kraftplattor. Grundläggande kunskaper inom mekanik och linjär algebra krävs för fullständig förståelse.

2.1 Segmentering och lokala koordinatsystem

Kroppen är en sammansättning av ben, muskler och senor som tillsammans utgör ett reglersystem som varje dag gör det möjligt för oss att utföra komplexa uppgifter. I en modellering av kroppen finns det alltså mycket att ta hänsyn till och vissa begränsningar måste därför göras för att möjliggöra en analys. I det här arbetet betraktas kroppen som en sammansättning av åtta stela segment; fötter, skenben, lår, bäcken och bröstkorg, sammanlänkade i leder. Anledningen till att det inte finns några segment definierade för armar och huvud är att dessa inte är intressanta för detta arbete.

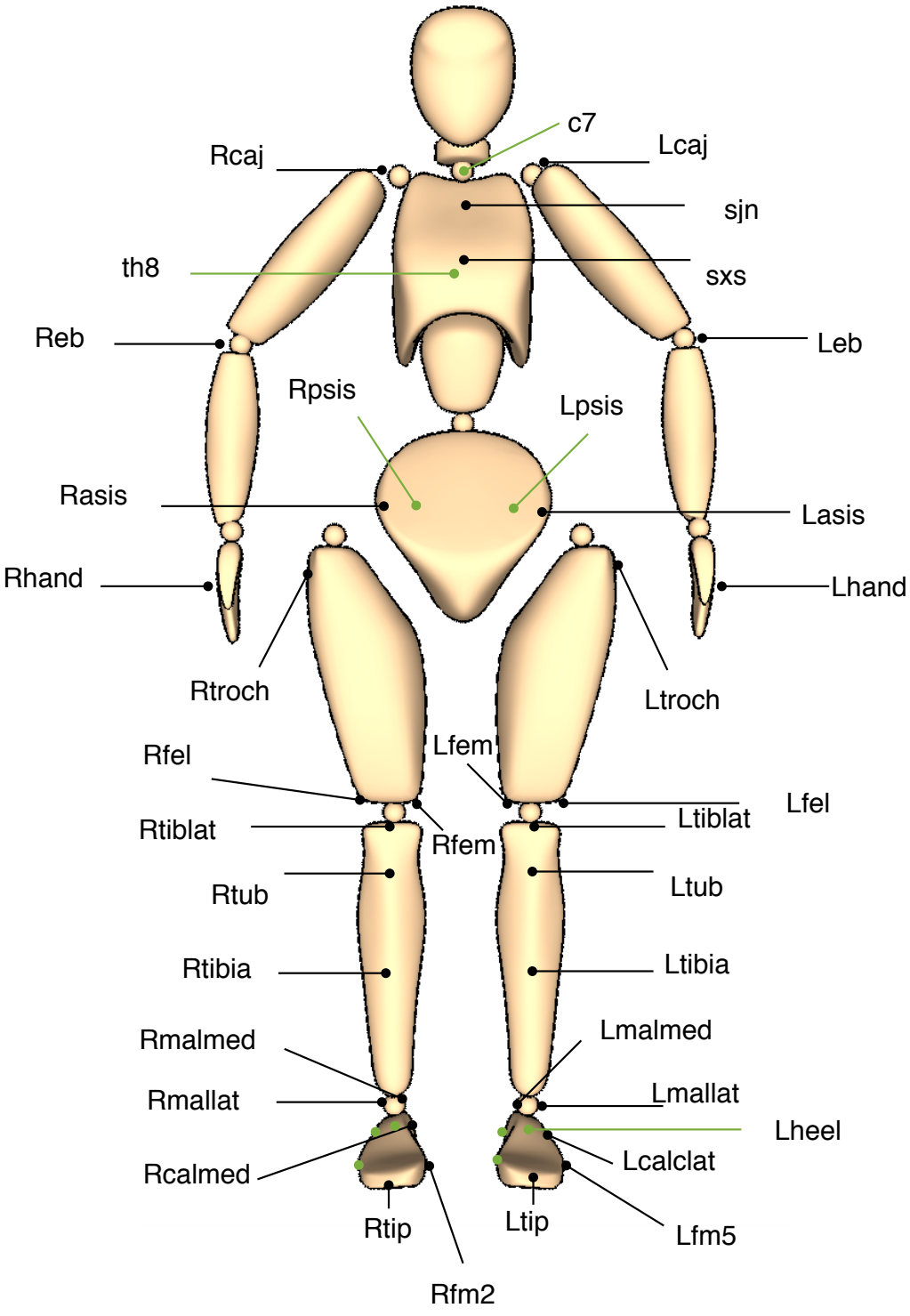
Ett segment definieras utifrån minst tre markörer som inte alla ligger längs samma linje på kroppsdelen som ska betraktas som stel, bild av marköruppsättningen som använts för projektet ses i figur 2.1. Varje segment har ett eget lokalt koordinatsystem med tillhörande origo som följer med segmentet i dess rörelse. Tanken är att den stela kroppen kan beskrivas fullständigt med hjälp av detta koordinatsystem, då segmentet rör sig så förflyttas det lokala koordinatisystemet på motvarande sätt. Koordinatsystemen är definierade så att ξ-axeln alltid är riktad åt kroppens högra sida, η-axeln rakt fram och ζ-axeln uppåt. ξ, η och ζ betecknar det lokala koordinatsystemets x-, y- respektive z-axel.

De två första axlarna för varje segments lokala koordinatsystem genereras utifrån markörerna som bygger upp det och den sista axeln fås av kryssprodukten mellan de två första enligt

$$\hat{\xi} \times \hat{\eta} = \hat{\zeta}, \tag{2.1}$$

eventuellt med cyklisk permutation. Vilken av axlarna som definieras först är egentligen godtyckligt men regeln med hur axlarna i stora drag ska ligga ska i slutändan alltid gälla. Observera att ekvation (2.1) endast är giltig då vektorerna är normerade.

En transformation av en koordinatpunkt $\mathbf{P}$ i det, i rummet fasta, globala koordinatsystemet till en punkt $\mathbf{P}'$ i ett av de lokala koordinatsystem fås av en linjärsamt en rotationstransformation. Den linjära transformationen sker via koordinat-



Figur 2.1: Uppsättning av markörer som har använts för att definiera kroppen i detta arbete. Totalt behövs 42 namngivna markörer för att kunna göra analysen. Notera att gröna streck till markörnamnen betyder att markören är placerad på baksidan av kroppen. ”R” och ”L” står för ”right” respektive ”left”.

vektorn för det lokala koordinatsystemets origo

$$\mathbf{O} = \begin{bmatrix} O_x \\ O_y \\ O_z \end{bmatrix} \tag{2.2}$$

och rotaionstransformationen via rotationsmatrisen

$$RM = \begin{bmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{bmatrix}. \tag{2.3}$$

Rotationsmatrisen är uppbyggd av det lokala koordinatsystemets komponenter uttryckt i det globala koordinatsystemet och roterar alltså det globala koordinatsystemet så att det får samma orientering som det lokala. Matrisen är ortonormal. Den totala transformationen mellan $\mathbf{P}$ och $\mathbf{P}'$ är alltså en sammansättning av (2.2) och (2.3) enligt

$$\mathbf{P}' = RM(\mathbf{P} - \mathbf{O}). \tag{2.4}$$

2.2 Vinklar mellan segment

För att studera hur de stela segmenten är vridna i förhållande till varandra, alltså hur ett lokalt koordinatsystem är orienterat jämfört med ett annat, betraktar vi Cardan-Euler-vinklar i en xyz-sekvens för segment som sitter ihop i en led. Ledvinklarna, som är av intresse, fås genom att det lokala koordinatsystemet för det ena segmentet i tre steg roteras till att ha samma orientering som det andra. Först en rotation med vinkeln α kring ξ, ξ_1-axeln, vilket gör att η- och ζ-axlarna får de nya riktningarna η_1 och ζ_1, sedan en vinkel β kring den nya η_1, η_2-axeln, vilket medför att ξ_1- och ζ_1-axlarna får nya riktningar ξ_2 och ζ_2, och till sist en rotation med vinkeln γ kring ζ_2, ζ_3-axeln till den nya riktningarna ξ_3 och η_3, detta illustreras i figur 2.2. Det lokala koordinatsystemet har alltså ändrat orientering från ξ, η, ζ till ξ_3, η_3, ζ_3 vilket fullständigt kan representeras av α, β och γ samt de initiala koordinatsystemet. Varje led har således tre frihetsgrader.

Vinklarna α, β och γ beräknas genom ekvationerna (2.5), (2.6) respektive (2.7) där R är en 3×3-matris som är en sammansättning av rotationsmatriserna för segmenten för vilka vinkeln mellan dem ska beräknas.

$$\alpha = \arctan\left(\frac{-R_{32}}{R_{33}}\right) \tag{2.5}$$

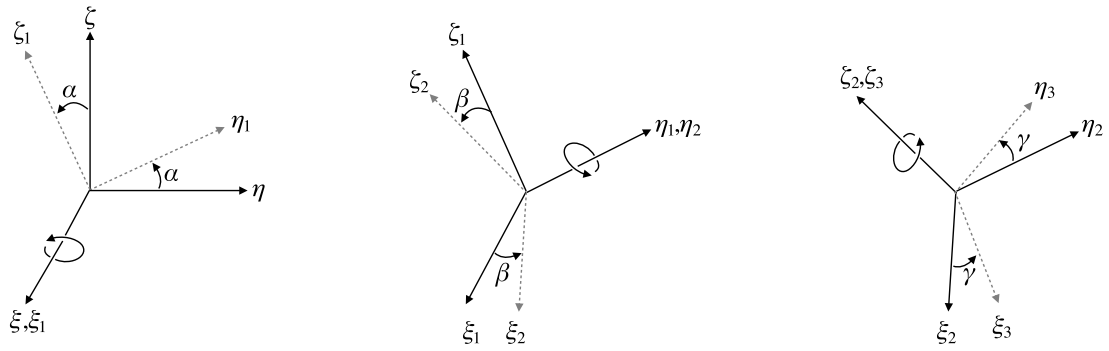
$$\beta = \arctan\left(\frac{R_{31}}{\sqrt{R_{11}^2 + R_{21}^2}}\right) \tag{2.6}$$

$$\gamma = \arctan\left(\frac{-R_{21}}{R_{11}}\right) \tag{2.7}$$

Matrisen R definieras enligt

$$R = RM_{\text{avlägsen}} RM_{\text{nära}}^T. \tag{2.8}$$

där $RM_{\text{avlägsen}}$ och $RM_{\text{nära}}$ är rotationsmatrisen för det segment som är längst bort respektive närmast centrum av kroppen, vilket är valt till bäckenet.



Figur 2.2: Rotation av lokalt koordinatsystem i xyz-sekvens av Cardan-Eulervinklar.

2.3 Masscentrum

För att betrakta kinematiken för en rörelse är det viktigt att välja en punkt på varje segment som är representativ för rörelsen av hela segmentet. Eftersom origo för respektive segments lokala koordinatsystem är definierat ganska godtyckligt så är detta tyvärr inte en lämplig punkt att välja. Istället definieras masscentrum för respektive segment och denna väljs till att representera hela segmentet, detta enligt tabell 2.1. Massan för varje segment kan därefter också bestämmas vilket kommer tillämpas i beräkning av krafter. Notera att denna modell för att beskriva kroppen är approximativ och att parametrarna i tabellen är empiriskt bestämda. Detta kan därför stämma mer eller mindre bra beroende på hur personen, som mätningarna gjorts på, är byggd.

Tabell 2.1: Segmentspecifika parametrar som är relevanta för denna modellering. Ändpunkterna följer beteckningarna på markörerna enligt figur 2.1. Notera att respektive segments delmassa inte summerar till 1. Värdena kommer från tabell 3.2 i [4].

Segment	**Beteckning för massan**	**Ändpunkter**	**Relativt avstånd ändpunkt–masscentrum**	**Segmentets massa/Total massa**
Fot	m_f	heel – tip	0.429	0.0145
Skenben	m_s	tub – mallat/malmed	0.433	0.0465
Lår	m_l	troch – fel/fem	0.433	0.1000

2.4 Hastighet och acceleration

När en punkt, för vilken kinematiken ska betraktas för, har definierats är det dags att beräkna hastigheten och accelerationen för punkten vid alla tidpunkter i mätningen. Lägeskoordinaterna, $\mathbf{x} = (x, y, z)$, för masscentrum av varje segment kan beräknas

utifrån datan från markörerna och hastighet och acceleration fås från första- respektive andraderivatan av dess position i rummet. Eftersom endast diskreta värden för att beskriva rörelsen tillhandahålls, alltså ej någon analytisk funktion, används här den numeriska finita differensmetoden för approximation av derivatorna. För att detta ska bli så stabilt som möjligt tillämpas alla de tre vanligaste formerna för denna approximation; framåtdifferens (2.9), bakåtdifferens (2.10) samt centraldifferens (2.11), för första- respektive andraderivatan.

$$\mathbf{v}_1 = \frac{\mathbf{x}_2 - \mathbf{x}_1}{\Delta t}, \quad \mathbf{a}_1 = \frac{\mathbf{x}_1 - 2\mathbf{x}_2 + \mathbf{x}_3}{(\Delta t)^2} \tag{2.9}$$

$$\mathbf{v}_n = \frac{\mathbf{x}_n - \mathbf{x}_{n-1}}{\Delta t}, \quad \mathbf{a}_n = \frac{\mathbf{x}_n - 2\mathbf{x}_{n-1} + \mathbf{x}_{n-2}}{(\Delta t)^2} \tag{2.10}$$

$$\mathbf{v}_i = \frac{\mathbf{x}_{i+1} - \mathbf{x}_{i-1}}{2\Delta t}, \quad \mathbf{a}_i = \frac{\mathbf{x}_{i+1} - 2\mathbf{x}_i + \mathbf{x}_{i-1}}{(\Delta t)^2} \tag{2.11}$$

Framåtdifferensen tillämpas endast på det första tidssteget, bakåtdifferensen på det sista och centraldifferensen på alla där emellan. Dessa numeriska metoder fungerar mycket bra så länge som datan som processeras inte innehåller alltför mycket brus. Om datan inte är helt glatt blir följdfelet som då uppstår på grund av instabilitet större för högre ordningens approximation av derivatan. Centraldifferensen är den av approximations metoderna som är stabilast, alltså ger minst fel, och det är därför lämplig att den används mest.

2.5 Krafter och impuls

För att räkna på vilka krafter som kroppen utsätts för under en rörelsesekvens används metoden för invers dynamik. Denna metod utnyttjar den kända kinematiken för att beräkna de minimala krafter som rörelsen ger upphov till. Förenklat kan man säga att vi löser Newtons andra lag i ekvation

$$\Sigma\mathbf{F} = m\mathbf{a} \tag{2.12}$$

med ett delvis okänt vänsterled.

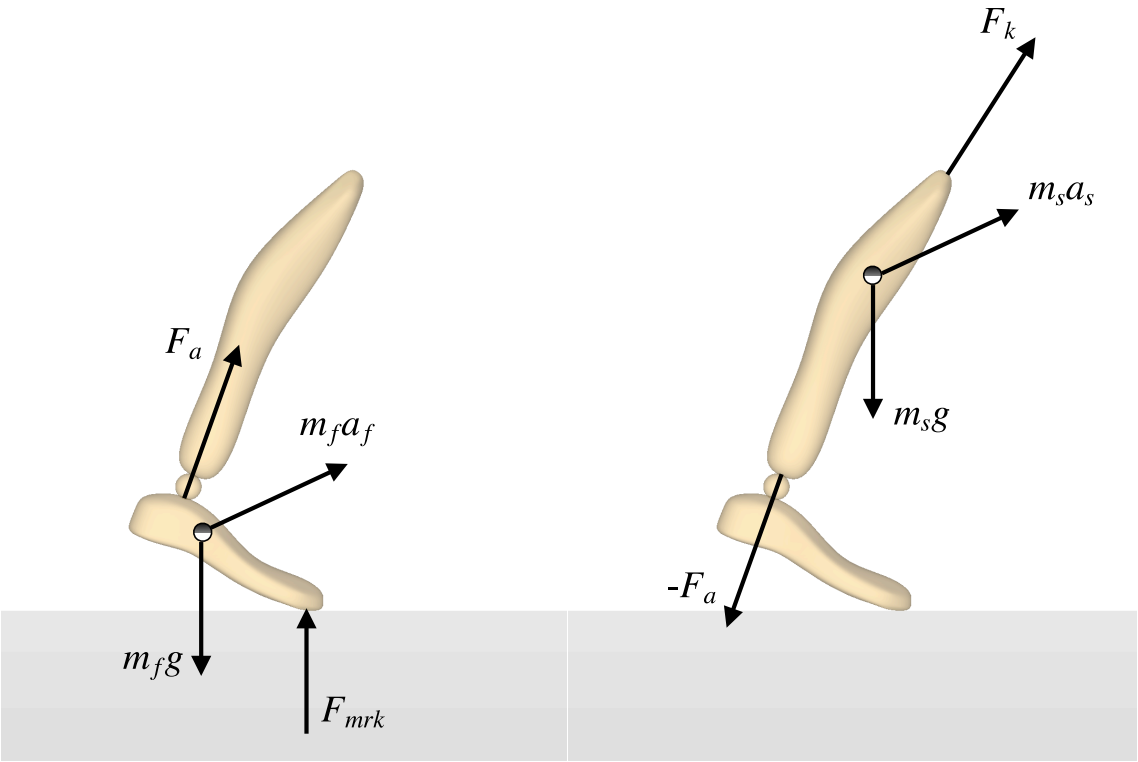
De krafter som vi ska studera i detta arbete är de som verkar på anklar och knän, alltså på lederna mellan de stela segmenten fot och skenben respektive skenben och lår. Detta eftersom det är där som det ofta uppstår belastningsskador. Efter en friläggning av foten fås att kraften på ankeln kan beräknas enligt

$$\mathbf{F}_a + \mathbf{F}_{mrk} + m_f\mathbf{g} = m_f\mathbf{a}_f \implies \mathbf{F}_a = m_f(\mathbf{a}_f - \mathbf{g}) - \mathbf{F}_{mrk} \tag{2.13}$$

där index a står för ankel, mrk för markreaktionskraft och f för fot. Här utnyttjar vi alltså att vi tidigare har definierat massor för segmenten samt beräknat accelerationen för dem. Markreaktionskraft är den kraften som kraftplattorna mäter.

Motvarande friläggning för skenbenet medför att kraften som verkar på knät fås som

$$-\mathbf{F}_a + \mathbf{F}_k + m_s\mathbf{g} = m_s\mathbf{a}_s \implies \mathbf{F}_k = m_s(\mathbf{a}_s - \mathbf{g}) + \mathbf{F}_a \tag{2.14}$$



Figur 2.3: Friläggning av fot till vänster respektive skenben till höger.

där index k står för knä och s för skenben. Bild av de båda friläggningarna finns i figur 2.3.

Beroende på i vilket koordinatsystem, globalt eller lokalt, som krafterna beräknats i kan det vara lämpligt att byta system för att lättare göra en analys av resultatet. En sådan transformation görs med hjälp av rotationsmatrisen enligt

$$\mathbf{F}_{\xi\eta\zeta} = RM\mathbf{F}_{xyz} \tag{2.15}$$

där krafter beräknade i det globala koordinatsystemet istället uttrycks i det lokala.

Då externa krafter, så som markreaktionskraften, är kända kan det vara intressant att beräkna vilken impuls, $\mathbf{I}$, som dessa utövar på kroppen. Impuls är vektoriell storhet med tolkningen ändring av rörelsemängd vilken beräknas som integralen av en kraft $\mathbf{F}$ över ett visst tidsintervall t_1 till t_2, detta enligt ekvation (2.16) [5]. Notera att vi kommer behöva integrera diskret för att beräkna impulsen och vi använder oss då av funktionen `trapz` i MATLAB.

$$\mathbf{I} = \int_{t_1}^{t_2} \mathbf{F} dt \tag{2.16}$$

3
Metod

I detta avsnitt behandlas de metoder som använts i denna rapport. Gruppen har huvudsakligen utgått från en metod med kvantifiering som tillämpats på en biomekanisk modell då den inversa dynamiken och kinematiken ska betraktas och undersökas för en innebandyspelare.

Det faktiska arbetet ligger till grund för denna rapports metod kan delas in i tre mindre delar; datainsamling, analys av data och framtagning av resultat. Det teoretiska underlag som presenterades i tidigare kapitel är det som tillämpats under den biomekaniska modelleringen.

3.1 Datainsamling

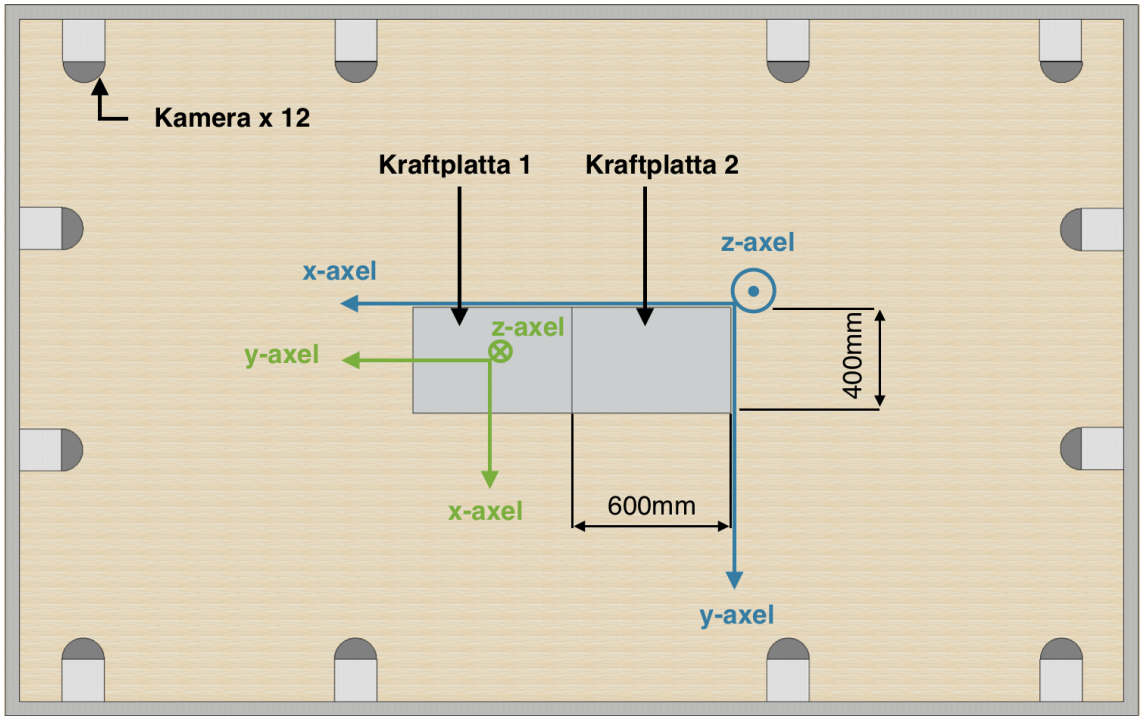
För att möjliggöra en datainsamling gjordes mätningar med endast en metod; rörelseanalyssystemet Qualisys Motion Capture. Datan samlades in vid två separata tillfällen i två olika laboratorier. Vid datainsamlingen gjordes både statiska och dynamiska mätningar. Första datainsamlingen gjordes för att ett program skulle kunna byggas i MATLAB. Det andra tillfället var då den faktiska datainsamlingen gjordes på tre testpersoner som spelade innebandy. Denna data användes sedan i analysen av data och framtagningen av resultat för att bestämma vad som skulle visualiseras och varför just detta, samt hur detta skulle göras på bästa sätt för att vara användbart för mottagaren.

3.1.1 Qualisys Motion Capture

Qualisys Motion Capture är ett program som använder rörelsedata för att studera och presentera mänskliga rörelser. Systemet kvantifierar rörelser precist och systemets flexibilitet gör att flera användningsområden inom biomekaniken är möjliga.

När en inspelningssekvens gjordes registrerade kamerorna positionen på markörerna som reflekterade och gav en datapunkt över vart markörerna befann sig i rum och tid gentemot det globala koordinatsystemet. Denna data kom sedan att användas för att möjliggöra biomekaniska beräkningar med hjälp av ett program som gjordes i MATLAB, vilket beskrivs senare i detta kapitel.

Kamerorna var uppsatta i en ram runt rummet och de två kraftplattorna som kom att användas satt i golvet mitt i rummet enligt figur 3.1. De infraröda kamerorna har en inspelningsfrekvens på 400 Hz och detsamma gäller för kraftplattorna.



Figur 3.1: Illustrativ bild över laboratoriet där datainsamlingen gjordes. Som bilden illustrerar finns det två olika koordinatsystem i miljön, ett globalt koordinatsystem som är det blå och ett lokalt koordinatsystem för kraftplattorna som är det gröna. En kraftplatta var 600 mm lång och 400 mm bred. Bilden över laboratoriet är inte skalenlig.

3.1.2 Qualisys Track Manager

Mjukvaran Qualisys Track Manager är ett program som används för att samla in data vid en inspelning med Qualisys Motion Capture. Systemet användes i detta fall för att samla in data i 3D i realtid. Den data som hade registrerats vid inspelningen gav varje markörer positionen i xyz-koordinater. I programvaran fanns även en modul, Automatic Identification of Markers, som automatiskt identifierar markörernas banor när de har blivit namnsatta och registrerade i programmet. När markörerna har blivit registrerade kunde det sedan fyllas i luckor i datan som kunde ha uppkommit om markörerna inte var synliga under hela inspelningen, detta gjordes med funktionen Gap fill trajectories. Denna funktion räknade ut hur banan förmodligen hade sett ut för markören om den inte hade lossnat från testpersonen eller inte registrerats av kamerorna under inspelningen.

Programvaran hade även funktionen att kunna ge data över tryckcentrum från kraftplattorna, var resultantkraftens angreppspunkt är, och de krafter som krafplattorna utsattes för under vändningen vilket kunde exporteras till en fil som sedan användes i MATLAB.

3.1.3 Statisk mätning

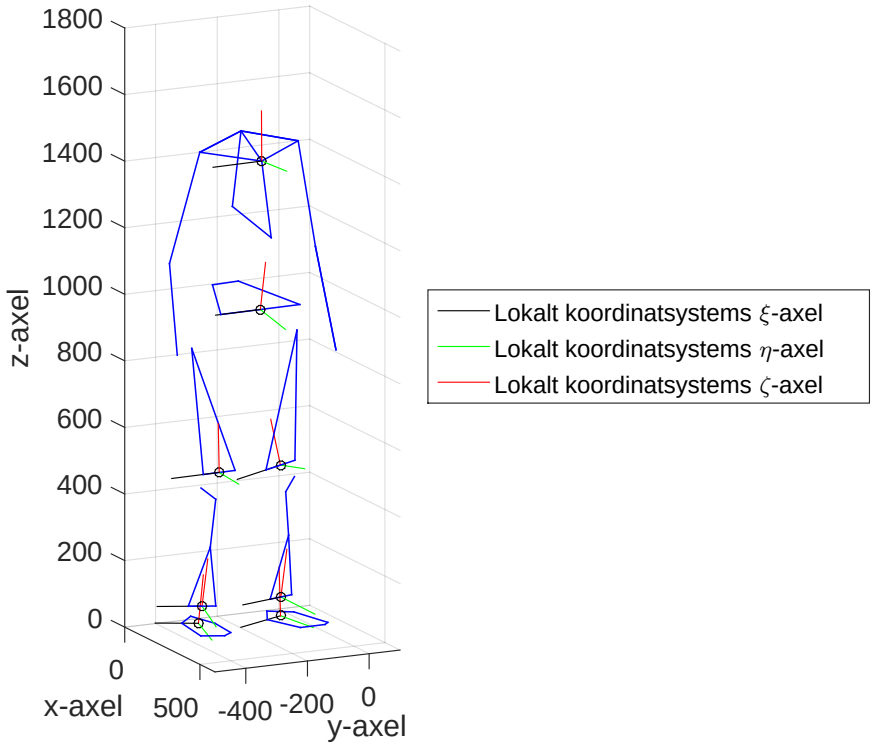
När en datainsamling gjordes startades det alltid med att markörerna sattes fast på testpersonerna efter uppsättningen som illustreras i figur 2.1. Därefter gjordes en statisk mätning där testpersonen fick stå normalt i mitten av rummet för att kamerorna skulle kunna registrera alla markörer som fanns på personen. Därefter definierades alla åtta segmenten och deras lokala koordinatsystemen, vilket visas i figur 3.2.

Den statiska mätningen ger en position för alla markörer gentemot varandra, vilket kan ses som ett jämviktsläge. Analysen på de statiska mätningarna som ger jämviktsläget görs för att senare kunna beskriva hur mycket olika vinklar mellan segmenten på testpersonen skiljer från rörelsesekvenserna.

3.1.4 Dynamisk mätning

Den dynamiska datan som tidigare mättes under inspelningarna var det som analyserades i nästa steg. Den första mätningen, där ett program skulle byggas, började med en gångsekvens dels för att testpersonen skulle få möjligheten att sikta in sig och träffa kraftplattorna dels för att till en början göra en enklare beräkning av en gångsekvens som det finns tidigare data från [4] att jämföra med. Därefter gjordes en sekvens där man springer och sist en vändningssekvens där de faktiska beräkningarna tillämpades i MATLAB.

Vid det andra datainsamlingstillfället som gjordes var det endast en vändningssekvens som studerades under inspelningen och samlades data ifrån. Detta var den data som senare kom att studeras i MATLAB och presenteras i resultatet.

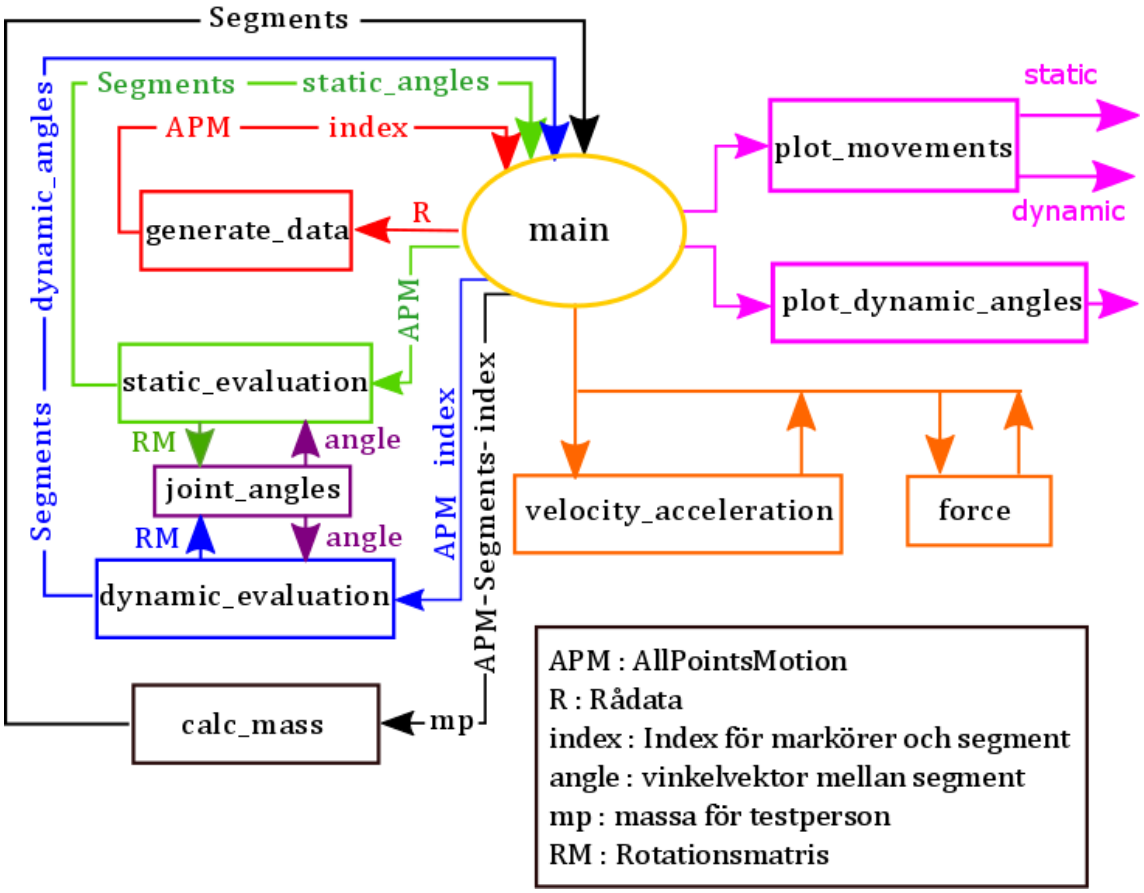


Figur 3.2: Bild från en statisk mätning där de lokala koordinatsystemen för varje segment illustreras.

3.2 Analys av data – MATLAB

För att analysera den insamlade datan från mätningarna har den behandlats i MATLAB. För att säkerställa att programmet klarar av de analyser som är tilltänkta behöver vissa krav på programmet uppfyllas. Programmet måste vara förhållandevis effektivt skrivet för att spara på datakraft och tillåta större analyser. Enbart en sekvens med en innebandyspelare som springer över en kraftplatta och vänder under två sekunder skulle ge 800 tidsteg för programmet att hantera då samplingsfrekvensen är 400 Hz, vilket Qualisys kameror under mätningarna har varit inställda på. Programmet måste klara längre sekvenser än detta och fortfarande vara effektivt. Programmet måste också vara generellt skrivet för att klara av att hantera data från flera olika mätningar med olika personer. Detta förutsätter dock att markeruppsättningarna på testpersonerna är samma och följer det standardiserade protokoll beskrivet i avsnitt 2.1. En annan utmaning blir att verifiera de beräkningar som utförts i programmet. Den verifieringsmetod som kommer tillämpas är att studera och noga granska framräknade resultat och bedöma om dessa är rimliga eller ej. Då numerisk data kan vara mycket svår att bedöma kommer all data visualiseras för att underlätta verifieringen.

Programmet som gruppen har arbetat med är uppbyggt och skrivet enligt figur 3.3. Huvudskriptet är döpt till `main` och det är därifrån all data utgår. Skriptet delegerar data till funktionsfiler som returnerar relevant beräkningsdata tillbaka till `main`-skriptet. Användandet av många funktionsfiler före en stor funktionsfil



Figur 3.3: Schematisk bild över programmet.

togs eftersom en stor funktionsfil som hanterar alla beräkningar skulle i längden vara ineffektivt. Ständiga ändringar och korrigeringar behövde göras under arbetes gång och att enkelt kunna lokalisera rörande funktion och korrigera denna utan att påverka resterande funktioner gjorde detta tillvägagångssätt mer fördelaktigt.

Programmet är uppbyggd i en hierarkiskt ordning och `main` är dess toppskikt. Filerna som ligger underordnande från `main` måste köras strikt i denna ordning eftersom efterföljande funktioner är beroende av den data som tidigare funktioner har beräknat. Funktionsfilerna som anropas från `main` är följande:

1. `generate_data`
2. `static_evaluation`
3. `dynamic_evaluation`
4. `calc_mass`
5. `velocity_acceleration`
6. `force`
7. `plot_movements`
8. `plot_dynamic_angles`

Fields	Field 1	Field 2	Field…	Field…	Field m-1	Field m
1	data	data	data	data	data	data
2	data	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
n-1	data	…	…	…	…	…
n	data	data	data	data	data	data

Figur 3.4: `Structure` med m fält och n rader. Fälten och raderna kan byggas på med ny data allt eftersom behov uppstår.

3.2.1 `generate_data`

När programmet körs från `main` öppnas den sparade filen från Qualisys med rådata och skickar in denna i `generate_data`. Funktionen genererar den data som behövs för vidare beräkningar och utelämnar annan information som inte behövs i programmet. Under arbetets gång har denna filen ändrats och korrigerats flera gånger, allt eftersom det krävts mer data och information till senare skrivna funktioner som inte har varit aktuella i början av projektet. Hanteringen av data och hur den sparas effektivt har därför varit viktigt eftersom mycket information ska kunna kommas åt smidigt till senare ekvationer. Att spara all data i flera olika variabler skulle ge ohanterligt många variabler och därför både vara ineffektivt och opraktiskt. Detta löstes genom att använda `structure` i programmet. En `structure` är en datatyp som grupperar data genom att använda sig av databehållare som kallas för fält, se figur 3.4. Varje fält kan innehålla all typ av data, till exempel en cell bestående av en matris eller en sträng med text. Mängden av olika typer av data som kan sparas gör användandet av `structure` väldigt flexibelt. All data kan enkelt kommas åt med hjälp av punktnotation på formen `structureName.fieldname`.

En av de första operationerna som `generate_data` utför är att jämna ut indatan och ta bort eventuella störningar som uppstått under insamlandet av data med Qualisys Motion Capture. Brus och andra störningar förekommer i datan och kan medföra fel i resultaten från beräkningarna. För att kompensera mot dessa störningar jämnas dessa ut med hjälp utav en inbyggd funktion i MATLAB som heter `smooth`. Funktionen använder sig av en metod som kallas för glidande medelvärde. En serie av medelvärden skapas genom att bilda medelvärdet av alla värden i ett visst tidsintervall. Medelvärdet tas från lika många punkter från varje sida av centralpunkten för att säkerställa variationerna i medelvärdet är pågrund av variationer i datan snarare än variationer i tiden. Glidande medelvärde definieras som

$$yy(t) = \frac{y(t-n) + y(t-n+1) + ... + y(t) + ... + y(t+n-1) + y(t+n)}{2n+1}$$

där yy är det nya störningskompenserade värdet av y, t den diskreta tidpunkten och $(2n+1)$ är spannet för det glidande medelvärdet. Notera att vid ändpunkterna, det vill säga när tiden t är nära början eller slutet av intervallet blir utjämningen inte lika bra eftersom det inte finns tillräckligt med tidspunkter för att upprätthålla spannet. Spannet kommer därför minska tills det endast innehåller en tidpunkt och

således jämna ut störningar sämre vid ändpunkterna. De kvarstående störningarna vid ändpunkterna ger dock inget betydande fel, men måste t.ex. kompenseras för när accelerationen beräknas i `velocity_acceleration` då den numeriska andraderivatan förstorar felet och ger felaktiga utslag i resultaten. Teorin bakom den numeriska andraderivatan och varför felet blir större beskrivs i avsnitt 2.4

När eventuella signalstörningar har blivit kompenserade går `generate_data` igenom varje markör, namnger den och sparar relevant data i en `structure` som namnges `AllPointsMotion`. Eftersom `AllPointsMotion` är en `structure` kan mer data enkelt sparas och läggas till allt eftersom behov uppstår. Den data som sparas i `AllPointsMotion` är följande:

1. `Header`: Varje markörs namn sparas som en sträng.
2. `PosTime`: Markörens position för varje tidssteg sparas. Om datan som genereras är statisk kommer ett medelvärde över alla tidssteg beräknas och sparas istället.
3. `Bodypart`: För att enklare hålla reda på var varje markör och dess plats på kroppen skapas `Bodypart` som delar upp kroppen i olika delar.
4. `PlotRank`: Definierar vilken ordning markörerna för varje `Bodypart` förhåller sig till varandra. Detta används senare till att plotta markörerna och dra streck mellan dem i rätt ordning.
5. `Force`: Om markören sitter på en kraftplatta har den även en kraft som definieras och sparas till `Force`.
6. `COP`: Centre Of Preassure förkortat `COP` är den punkt på kraftplattorna som tar upp den största tryckkraften från foten.
7. `norm_Force`: Den euklidiska normen av kraften.

För att kunna komma åt rätt markör i `AllPointsMotion` vid senare beräkningar utan att behöva skriva ned dess värde skapas ett index åt markörerna och markörens namn mappas med dess rad i `AllPointsMotion`. På så sätt kan respektive markör kallas på med hjälp av deras namn eftersom namnet är kopplat till dess position. När programmet har körts returneras `AllPointsMotion` tillsammans med `index` tillbaka till huvudskriptet `main`. `generate_data` är skriven på ett sådant sätt att funktionen kan hantera och generera både statisk och dynamisk data. I efterföljande text kommer därför den statiska datan i `AllPointsMotion` refereras som `S_AllPointsMotion` och den dynamiska som `D_AllPointsMotion`. I figur 3.3 representeras `generate_data` av den röda slingan. Den fullständiga funktionen återfinns i Appendix B tillsammans med resterande funktioner där exakta beräkningar och operationer kan läsas.

3.2.2 `static_evaluation`

Den gröna slingan i figur 3.3 visualiserar funktionsfilen `static_evaluation`. Denna funktionsfil utför beräkningar på den statistiska data som sparats i strukturen `S_AllPointsMotion`, tidigare genererad i funktionen `generate_data`. Eftersom

det inte är intressant att betrakta markörerna var för sig definieras olika segment som byggs ihop genom att kombinera markörer tillsammans. Totalt definieras åtta segment, vilka utgör representationer för höger och vänster fot, höger och vänster skenben, höger och vänster lår, höft och bröstkorg, i enlighet med avsnitt 2.1. Varje segments lokala koordinatsystem definieras och segmenten samlas i strukturen `S_Segments` där följande data har beräknats och sparats:

1. `Part`: Varje segments namn.
2. `Origin`: Segmentets origo.
3. `X`: (x, y, z) värden för segmentets X-axel.
4. `Y`: (x, y, z) värden för segmentets Y-axel.
5. `Z`: (x, y, z) värden för segmentets Z-axel.
6. `RM`: Rotationsmatrisen för segmentet.

I `static_evaluation` anropas även en annan funktionsfil, `joint_angles`, där de statiska vinklarna beräknas med hjälp utav segmentens rotationsmatriser. Den mörklila funktionen i figur 3.3 visar hur funktionerna integrerar med varandra. Dessa vinklar, `static_angles`, är vinklarna för segmenten när testpersonen står vanligt och blir det som kommer att användas som jämviktsläge för att beräkna de dynamiska vinklarna i `dynamic_evaluation`. De statiska segmenten `S_Segments` returneras tillsammans med `static_angles` tillbaka till `main`-skriptet.

3.2.3 `dynamic_evaluation`

När den statistiska datan har beräknas körs den dynamiska datan i `dynamic_evaluation`, vilket representeras av den blå slingan. Funktionen skiljer sig inte mycket från `static_evaluation` beräkningsmässigt. De dynamiska segmenten `D_Segments` definieras på samma sätt som `S_Segments` för den statiska datan, men med den direkta skillnaden antal tidssteg som beräknas. `static_evaluation` utför beräkningar med ett medelvärde över alla tidssteg och `dynamic_evaluation` beräknar med alla tidssteg för markörerna. De dynamiska vinklarna beräknas genom att först skicka in de dynamiska rotationsmatriserna i funktionsfilen `joint_angles`. Skillnaden mellan `static_angles` och de framräknade vinklarna från `joint_angles` utgör `dynamic_angles`.

3.2.4 `calc_mass`

Denna funktion beräknar masscentrum för segmenten enligt tabell 2.1 och representeras av den svarta slingan i den schematiska figuren. Segmentets totala massa sparas fältet `Segments.Mass` och dess masscentrum sparas in i `Segments.COM` där fältet `COM` står för Centre Of Mass.

3.2.5 `velocity_acceleration`

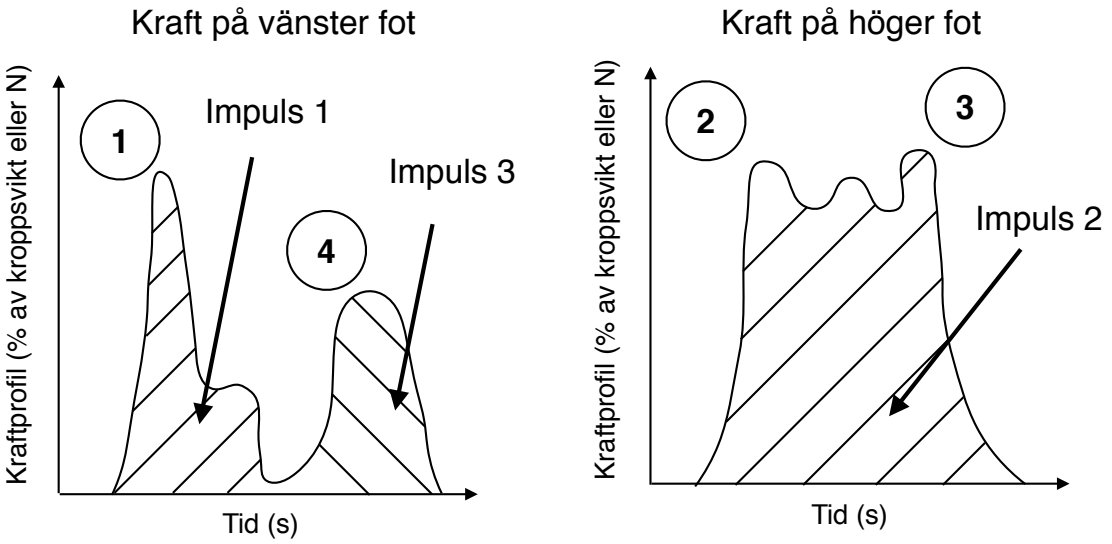
`velocity_acceleration` som representeras som en av de orangea slingorna i figuren 3.3 utför beräkningar på masscentrum av varje segment med syfte att beräkna hastighet och acceleration. Genom att tillämpa centraldifferens i enlighet med (2.11) kan första- respektive andraderivatan av markörernas position i rummet approximeras för varje tidssteg. Programmet sparar hastigheten och accelerationen som två fält i strukturen `D_Segments` som `velocity` och `acceleration`. Hastigheten och accelerationen som beräknats är vektorer med både storlek (magnitud) och riktning vilket gör det svårt att enkelt visualisera dessa på ett sätt som är lätt att förstå. En mer lämplig storhet är fart som till skillnad från hastighet saknar riktning och kan visualiseras i en tvådimensionell plott med längd över tid. Av den anledningen räknas farten ut genom normen av hastigheten och plottas över respektive segment. Av samma anledning beräknas även normen av accelerationen.

3.2.6 `force`

I funktionsfilen `force` beräknas krafterna som segmenten utsätts för under rörelsesekvensen som testpersonen utför. Eftersom det är två kraftplattor som uppmäter krafterna som testpersonen utövar under mätningarna finns också två olika kraftmätningar tillgängliga i `D_AllPointsMotin`, en från vardera kraftplatta. Den första operationen som `force` utför är att lokalisera vilken fot som träffar vilken kraftplatta för på så sätta koppla rätt krafter till rätt fot. Detta görs genom att identifiera vilken fot, över alla tidssteg, som har det kortaste avståndet till respektive platta. När det har fastställts beräknas all kraftpåverkan på respektive segment genom att använda tidigare uträknad massa för segmenten i fältet `D_Segments.Mass` och acceleration i fältet `D_Segments.acceleration` och tillämpa Newtons andra lag i enlighet med ekvation (2.12). Fälten som adderas till `D_Segments` är följande.

1. `force`: Den kraftvektor som utövas över segmentet.
2. `norm_force`: Normen av kraften `force`.
3. `Point`: Den angreppspunkt i vilken kraften verkar på segmentet. Denna punkt kommer vara där kraften `force` utgår från när kraften visualiseras i den tredimensionella plotten i funktionen `plot_movements` där både storlek och riktning visas.
4. `local_force`: Transformation från globalt koordinatsystem till lokalt görs med hjälp av rotationsmatrisen enligt ekvation (2.15) och sparas i fältet `local_force`.

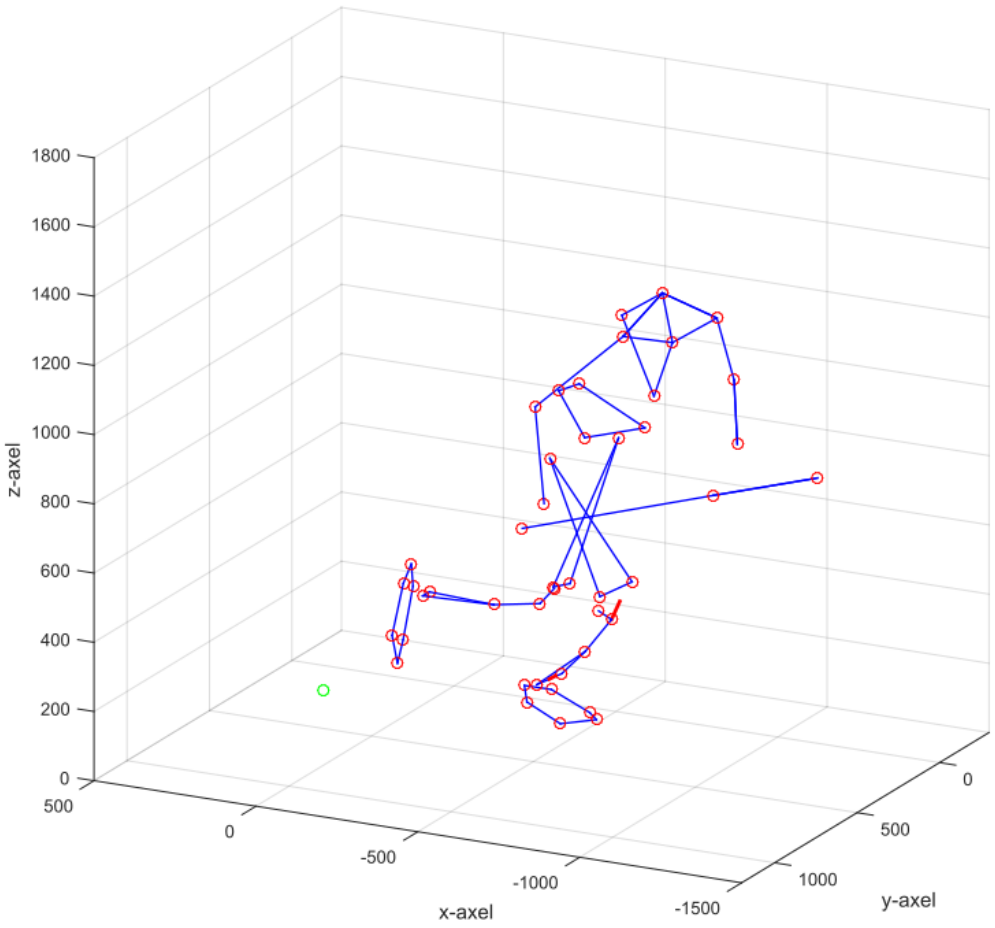
I funktionen beräknas också impulserna under vändningen. Impulsen beräknas med hjälp av MATLAB inbyggda funktion `trapz` som approximerar integralen av kraften enligt ekvation (2.16). Integralens gränser bestäms manuellt av användaren genom `ginput` i MATLAB. `ginput` är en grafisk input som tillåter användaren att bestämma start och slutpunkt på en graf genom att klicka med muspekaren. Start och slutpunkt väljs enligt figur 3.5 för att impulsen ska beräknas och returneras i kommandofönstret.



Figur 3.5: Illustration av en typisk kraftprofil i mätningarna som utförts. Denna bild fungerar som en förklaring till beteckningarna införda i tabell 4.1. Notera att siffrorna ①, ②, ③ och ④ markerar toppvärden av kraftresultanten och att impulsen beräknas i tre separata områden.

3.2.7 `plot_movements`

`plot_movements` visualiserar given data i form av en 3D-plot. 3D-plotten kan till exempel visa när testpersonen springer genom att plotta för varje tidssteg alla markörer, linjer emellan markörerna, de lokala koordinatsystemen och kraftkomponenterna på anklar och knän. Genom att studera dessa plottar efter varje ny beräkning har eventuella fel kunnat lokaliserats och korrigerats då dessa enkelt har visat sig i plotten. Under arbetets gång har `plot_movements` utökats för att kunna hantera mer data och korrigerats för att bli mer effektiv. I figur 3.6 visas en stillbild över en testperson med alla markörer utmärkta som röda ringar och blå streck emellan dem som tillsammans utgör varje kroppsdel. Hade en markör varit feldefinierad hade detta tydligt visat sig i plotten och på sätt säkerställs att datan genererats korrekt och lästs in riktigt. Användandet av `structure` har även här varit mycket användbart då data som i beräkningar varit irrelevanta, men som kan användas för att underlätta visualiseringen också har kunnat läggas in i strukturen och på så sätt inte ökat mängden variabler som indata till funktionen. `PlotRank` och `Bodypart` är två fält i `D_AllPointsMotion` som endast används i `plot_movements` med syftet att förenkla visualiseringen. För att kunna plotta upp all den mängd data som tagits fram effektivt har `plot_movements` eftersträvat att vara så generellt skrivet som möjligt för att kunna hantera alla olika mätningar utan att behöva korrigeras i funktionsfilen. Det är av den anledningen som programmet har så många in-variabler för att kunna styra vilken typ av visualisering som ska visas. Totalt är det fem olika in-variabler som, beroende vilken som sätts aktiv, visar olika delar av plotten. Det som går att ställa in är om det är en dynamisk eller statisk mätning som ska analyseras, om markörerna ska visas, om linjerna som definierar varje kroppsdel ska visas, om de lokala koordinatsystemen för segmenten ska visas och om kraften i anklar och knän

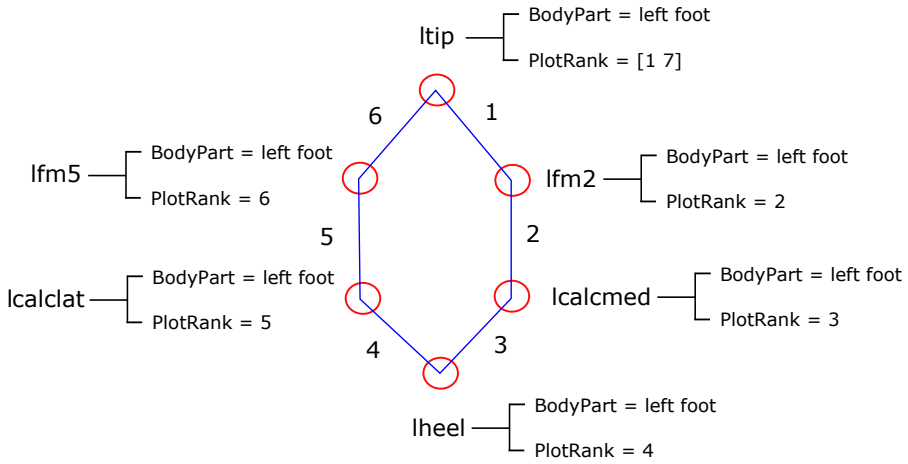


Figur 3.6: 3D-plot från `plot_movements`

ska visas i plotten.

Med hjälp av `structure` kan stor mängd data plottas upp med mycket lite kod genom en slinga som itererar genom strukturen och plottar upp datan för varje fält. Testpersonens markörer är ett sådant exempel. Varje markör har tre givna koordinater för varje tidsteg $[x, y, z]$ och har tidigare i `generate_data` genererats och sparats in i `AllPointsMotion.PosTime`. Funktionen `plot_movements` plockar ut markörernas koordinater och plottar upp dem en i taget till röda cirklar genom att iterera över alla markörer för varje tidssteg. Observera att det här inte spelar någon roll i vilken ordning markörerna plottas då de är oberoende av varandra.

Problem uppstår dock vid de tillfällen då den data som ska plottas upp är beroende av annan data. Linjerna mellan markörerna som definierar en kroppsdel är ett sådant exempel. Att på samma sätt plotta dessa är inte möjligt då strecken skulle gå kors och tvärs beroende på vilken ordning de ligger i `AllPointsMotion.PosTime` eftersom MATLAB opererar rad för rad. Strecken måste gå i en viss ordning om de ska bilda varje kroppsdel korrekt. Det är nu som de två fälten `Bodypart` och `PlotRank` i `AllPointsMotion` blir användbara. `Bodypart` tilldelar varje markör en kroppsdel och `PlotRank` vilken ordning markören ska plottas. Figur 3.7 visar sche-



Figur 3.7: Schematisk bild över ”left foot”.

matiskt hur programmet utför denna plottning över vänster fot som ett exempel. Vänsterfotens markörupsättning består av sex markörer ”ltip”, ”lfm2”, ”lcalcmed”, ”lheel”, ”lcalclat” och ”lfm5”. Dessa är tilldelade benämningen ”left foot” i fältet `BodyPart` och en given plottningsordning i fältet `PlotRank`.

En av de första operationerna `plot_movements` utför att att samla de koordinater som ska plottas och spara in dem i lokala variabler. Detta sker genom att programmet itererar över fältet `BodyPart` och söker efter markörer som tillhör respektive kroppsdel. När programmet har funnit en markör som tillhör en viss kroppsdel, i detta exempel ”left foot” i `BodyPart` sparas denna markör in i en lokal variabel kallad `coord_left_foot` på den plats som är given i `PlotRank`. På så sätt kommer alla markörers koordinater i rätt ordning i förhållande till varandra i `coord_left_foot` som blir en 7×1 matris eftersom sju är det högsta värde `PlotRank` antar. När `plot_movements` sedan plottar linjerna för vänsterfoten bildas streck 1 genom att ta första radens koordinater i `coord_left_foot` och binda samman den med den andra radens koordinater. Programmet fortsätter att binda samman markörerna på samma sätt i den ordning de ligger i listan tills alla koordinater har itererats. Notera att ”ltip” har två värden och således förekommer två gånger, både som det första värdet och som det sista. Anledningen till att ”ltip” tilldelats två värden är för att kunna sluta slingan genom att låta sista strecket, streck 6, gå tillbaka till ursprungspoitionen. Samma procedur utförs på resterande kroppsdelar tills alla är plottade. För att få en rörlig animation pausar `plot_movements` plotten kort innan nästa tidssteg plottas. Pausen i kombination med ett snabbt program får datan som visas att se ut att röra på sig.

3.2.8 `plot_dynamic_angles`

Denna funktion visualiserar de dynamiska vinklarna `dynamic_angles` för segmenten som beräknats i `dynamic_evaluation`. Funktionen är endast visuell och returnerar ingen data till `main`-skriptet utan plottar vinklarna mellan respektive segment och hur segmentets vinkel ändras i förhållande till respektive närliggande segment under hela rörelsesekvensen.

3.3 Framtagning av resultat

I detta arbete gavs resultatet i numeriska värden efter att datainsamlingen som gjordes applicerades i MATLAB. Det var viktigt att visualisera resultaten som hade beräknats på ett sådant sätt att de enkelt skulle kunna uppfattas och visa vad innebörden med mätningarna var. För att enklare kunna ta till sig informationen gjordes grafer för resulterande krafter i anklar och knän, markreaktionskrafter på fötter, lokala krafter som verkar på anklar och knän, vinklarna för segmenten i förhållande till varandra samt segmentens fart med hjälp av MATLAB. I dessa grafer går det att läsa av värdena för hur spelarnas rörelsemönster gav utslag på leder i kroppen, hur mycket spelarna bromsade in och tryckte ifrån med på kraftplattorna och vridningar för segmenten i förhållande till varandra.

Vid datainsamlingen som gjordes gavs alla mätningar i 3D-data med lägeskoordinaterna i x, y och z i det globala koordinatsystemet. För att möjliggöra en visualisering av resultatet i 2D-grafer behövdes storleken av resultanten beräknas. Detta för att det är enklare att förstå en längd i en 2D-graf än en vektor i rummet. Storleken fås av en euklidisk norm.

Vid framtagningen av kraftplottarna valdes att ställa en del tid gentemot procent av kroppstyngd och en del för tid gentemot resulterande kraft. Att ställa tid i förhållande till procent av deras kroppstyngd gjorde att det blev lättare att jämföra hur stora skillnaderna var mellan olika testpersoners krafter på plattan, i ankel och i knä.

4

Resultat

Då mätningar har genomförts och datan från dessa analyserats fås en mängd olika resultat där det som anses vara relevant presenteras i tabell 4.1. Dessa värden går att utläsa från graferna i figur A.1, A.2, A.3 samt A.4 i appendix A med tolkningen som ges av figur 3.5. Spelarna är, i dessa grafer, färgkodade för att underlätta jämförelse mellan dem. I alla mätningar som betraktats väljer spelarna att vända på samma sätt, de sätter vänster fot på första kraftplattan i löpriktningen och höger fot på den andra. Detta gör att jämförelse dem emellan underlättas ytterligare och att det blir meningsfullt att studera krafterna som spelarnas leder utsätts för i deras respektive lokala koordinatsystem.

Spelare Röd var den spelare som var snabbast på att utföra den studerade rörelsen och hon var också den spelare som hade den absolut största markreaktionskraften i sista steget innan vändingen. Hennes kropp utsätts därmed med mycket större krafter än de andra spelarna. Spelare Grön var nästan lika snabb men genererar inte alls lika stora markreaktionskrafter.

Vinklar mellan de olika segmenten under hela rörelsesekvensen för respektive spelare återfinns i figur A.5, A.6 och A.7 i appendix A. Dessa grafer presenterar alltså hur segmenten är vridna i förhållande till varandra vilket kan vara viktigt i en skaderiskanalys. Tillsist presenteras också en graf över farten, för respektive spelare under mätningarna, i figur A.8. Denna är färgkodad precis som de andra graferna och man kan där se antydningar till att spelarna har lite olika rörelsemönster; Spelare Grön har högre frekvens på kurvorna som representerar farten för fötterna än de andra spelarna och har alltså en ”trippande stil” innan vändningen.

All kod som ligger till grund för resultaten presenterade i det här kapitlet finns i appendix B. Arbetet har till mycket stor del gått ut på att bygga upp denna kod som tar fram en metod för den här typen av kvantitativa undersökningar och den kan därför ses som ett resultat i sig.

Tabell 4.1: Resultaten från mätningarna som är visualiserade i appendix A. Två olika tider för rörelsesekvensen har uppmäts, den första är tiden som en spelare har kontakt med någon av kraftplattorna och den andra är för ett givet intervall 2 m innan och 2 m efter första kraftplattan. Siffrorna ①, ②, ③ och ④ markerar toppvärden av resultanten för respektive kraft och att impulsen beräknas i tre områden, se figur 3.5 för vidare tolkning av hur dessa tas fram. Notera att LKS står för lokalt koordinatsystem.

		Spelare Blå	**Spelare Röd**	**Spelare Grön**
Längd (cm)		172	178	173
Vikt (kg)		71	70	73
Δt på plattor (s)		1.00	0.90	0.91
Δt hela vändningen (s)		1.89	1.83	1.84
Total markreaktionskraft	①	109.5	233.2	156.5
(% av kroppstyngd)	②	160.2	169.2	159.3
	③	160.5	158.4	166.4
	④	169.9	144.2	150.3
Resulterande kraft på	①	110.8	236.8	159.2
ankel (% av kroppstyngd)	②	161.9	170.6	161.1
	③	161.7	159.7	167.8
	④	171.4	145.7	151.6
Resulterande kraft på	①	114.8	248.6	167.5
knä (% av kroppstyngd)	②	167.9	176.3	167.4
	③	165.8	164.7	172.5
	④	177.8	151.3	156.9
Total impuls från	1	166.8	192.7	160.4
markreaktionskraft (Ns)	2	429.4	357.9	414.3
	3	250.4	213.4	274.1
Maximal kraft på vänster	ξ	161.5	694.9	430.7
ankel i LKS (N)	η	409.9	933.2	514.4
	ζ	1184	1305	957.4
Maximal kraft på vänster	ξ	153.3	354.4	148.6
knä i LKS (N)	η	876.8	683.4	787.7
	ζ	866.9	1629	1030
Maximal kraft på höger	ξ	354.7	373.8	322.0
ankel i LKS (N)	η	397.2	585.1	409.8
	ζ	1102	1045	1162
Maximal kraft på höger	ξ	224.0	259.7	135.9
knä i LKS (N)	η	695.1	664.3	701.2
	ζ	986.4	940.2	1073

5
Diskussion

Syftet med det här projektet var att ta fram och analysera kvantitativa skillnader mellan olika innebandyspelare för att kunna dra slutsatser om varför en del har ett bättre rörelsemönster än andra. De spelare som vi har analyserat var dock väldigt jämbördiga ifråga om tiden det tog för dem att utföra den valda sekvensen av rörelse och det är därför svårt att skilja dem åt för att identifiera hur en bra vändning ska utföras. Vi kommer i det här kapitlet att ta upp:

- Begränsningar med biomekanisk modellering
- Diskussion kring resultaten
- Rekommendationer till framtida projekt

5.1 Begränsningar med biomekanisk modellering

För att det ska vara möjligt att göra överhuvudtaget kunna göra en analys av den inversa dynamiken för kroppen behöver man göra ett antal antaganden. Det som är störst och som vi kommer diskutera här är att kroppen kan beskrivas av sammanlänkade stela segment. Detta är, som sagt, ett nödvändigt antagande men som tyvärr kan leda till störningar. Till exempel foten består av en mängd olika delar och kan därmed böjas på många olika sätt vilket alltså inte reflekteras i stelkroppsapproximationen. Många viktiga funktioner som foten har missas alltså och skaderisker för detta segment kan inte heller tas i beaktning.

Ett annat exempel på begränsningen med stela segment är att hänsyn inte tas till att den mjuka vävnaden, som muskler och hud, kan röra sig i förhållande till benen inuti kroppen. Detta medför, eftersom markörerna sitter fast på huden och i vissa fall också ovanpå kläder, att det stela segmentet kan ändra både storlek och interna egenskaper som position för masscentrum. I enkel kontroll kunde vi konstatera att masscentrums avstånd för skenbenet till en punkt på ankeln maximalt avvek 5 mm från en medelvärde av detta avstånd.

5.2 Diskussion kring resultaten

De spelare som deltagit i studien var, som tidigare påpekats, väldigt jämbördiga ifråga om snabbhet för den valda rörelsesekvensen men också ifråga om fysisk storlek även om de sinsemellan var olika muskulösa. Det är stora skillnader i de uppmätta värdena för storleken av markreaktionskraften i (1) vilket troligtvis beror på hur

mycket spelarna bromsar in i steget precis innan första kraftplattan och alltså hur mycket de måste bromsa in på kraftplattan för att överhuvudtaget kunna genomföra vändningen. Man kan tänka sig att det är mer skonsamt att de bromsande krafterna är utspridda över flera steg men att det inte är det som är mest tidseffektivt eftersom farten då successivt minskas. I våra mätningar har Spelare Röd, som är snabbast, mycket högre markreaktionskraft vid (1) än de andra och därmed en stor inbromsning precis innan vändningen medan Spelare Grön är nästan lika snabb men inte alls belastar sin kropp på samma sätt. De har alltså olika sätt att vända på som resulterar i samma effektivitet.

Ofta antas att då en kropp utsätts för en hög impuls innebär det, eftersom impuls är en ändring av rörelsemängd, att kroppen får hög hastighet vilket skulle medföra en snabb vändning. I de fall som studerats här stämmer dock inte det, Spelare Röd har den minsta impulsen under hela förloppet och att impulsen för de andra två spelarna är ungefär lika stor. En förklaring till detta är att vi inte tar hänsyn till riktningen då vi beräknar impulsen och för att den ska vara optimal ska den peka i löpriktningen.

Vid analys av reaktionskrafterna som andel av kroppstyngden visade sig dessa vara mindre än förväntat. Vid vanlig rak löpning över en kraftplatta har vi mätt upp en maximal markreaktionskraft på 180% av kroppstyngden vilket alltså jämförs med $109.5 - 233.2\%$ av kroppstyngden vid vändningen. Från början förväntades mycket högre krafter i vändningen än vid rak löpning men så är följaktligen inte fallet. Att det är hög risk att skada sig i en vändning kan då antas bero på att krafterna, till exempel i knät, tas upp snett och om kraften i det lokala koordinatsystemets ξ-led blir stor i kombination med stora rotationer kring η- och ζ-axlarna gör detta att skaderisken blir förhöjd.

5.3 Rekommendationer till framtida projekt

I detta arbete var målet att komma fram till en metod som i framtiden skulle kunna användas för att göra statistiska mätningar och studera innebandyspelares rörelsemönster. För att detta ska vara möjligt är det viktigt att försöka ha en större variation på de testpersoner som mätningarna görs så att en bredd på olika typer av spelare med olika rörelsemönster studeras. Dessutom fanns tyvärr inte möjligheten att utföra tillräckligt många mätningar för att få bättre variation och underlag då de lokaler och den utrustning som skulle behöva användas inte var tillgängliga efter gruppens behov. Därför skulle det vara att föredra att en bättre kontakt skulle finnas mellan kandidatgruppen, Chalmers och ett av laboratorierna där mätningar skulle kunna göras när det behövdes.

Under mätningarna som gjordes hände det att markörer åkte av och vid en sådan situation skulle det behövas göras en ny statistisk mätning men på grund av tidsbrist gjordes inte några nya. Att försöka hitta en tejp som sitter lika bra på både kläder som hud skulle vara att föredra. För övrigt så användes inte samma lokaler vid de olika mättillfällena vilket medförde att markörupppsättningen kan ha varit annorlunda tillfällena sinsemellan då det var olika personer som satte på markörerna.

Vidare borde även möjligheten att använda sig av fler kraftplattor än två ses över vilket skulle kunna hjälpa till med uppfattningen av hur inbromsningen och

accelerationen före respektive efter vändningssekvensen påverkar, samt att öka ytan som vändningen sker på. Detta eftersom Spelare Grön använde en större yta när vändningen gjordes då denna spelare svängde av och försökte få med sig farten som denne kom in med i vändningen. Spelare Röd och Spelare Blå gjorde en mer rak vändning vilket kanske är mer påfrestande för lederna då större krafter, i samband med högre vridningar mellan segmenten inverkar på lederna. Det skulle dessutom vara intressant att mäta över längre sekvenser då en längre sekvens skulle kunna ge mer data att analysera. Det måste också tas i beaktning att de kraftplattor som uppmätte krafterna i vändningen har lägre friktion än golvet som spelarna normalt utför vändningarna på. Detta medförde att de gled längs med plattorna i vändningen och kraften som uppmättes blev inte lika hög som den skulle ha blivit vid ett golv med mer friktion. Att använda skor som lämpar sig bättre kontra ytan mot kraftplattorna för att minimera glidande i vändningen är att föredra för att uppnå mer korrekta reslutat.

För att förbättra analysen av skaderiskerna samt kunna få en förståelse för vad det är som gör en person mer explosiv än en annan vid rörelser som dessa hade det varit av intresse att ha kontakt med personer som har kompetens inom ett sådant område. Förslagsvis skulle det kunna vara studenter på Idrottshögskolan som studerar idrottsmedicinska ämnen eller liknande.

6

Slutsats

I detta arbete har syftet varit att ta fram en kvantitativ metod för att studera kvinnliga innebandyspelares rörelsemönster. Metoden gör att det går att se skillnader på rörelsen hos olika spelare i form av krafter, vinklar mellan segment och hastigheter. För att kunna definiera vad som är en tidseffektiv och skademinimerande vändning behövs statistisk, alltså fler mätningar på fler spelare. Spelarna som mätningarna gjordes på i detta projekt var väldigt lika sett till fysisk storlek, dock hade de olika rörelsemönster.

Under mätningarna som gjordes på de tre spelarna konstaterades att skillnaderna i markreaktionskrafterna mellan de olika spelarnas vändningar var ganska stora. Dock är detta kanske inte den största anledningen till att den enes rörelsemönster är bättre än den andra. Spelare Röd som var snabbast var också den som hade den största markreaktionskraften vid inbromsningen. Spelare Grön har en väsentligt lägre markreaktionskraft men var i stort sett lika tidseffektiv under vändningen som studerades.

Slutsatsen av denna undersökning är att mätningarna som gjorts inte givit ett tillräckligt underlag för att några slutsatser med statistisk grund ska kunna dras om hur olika krafter påverkar rörelsemönstret eller skaderisken. Däremot kan det bekräftas att metoden som tagits fram skulle kunna användas för att analysera kvinnliga innebandyspelares rörelsemönster.

Referenser

[1] K. Pasanen et al., ”Neuromuscular training and the risk of leg injuries in female floorball players: Cluster randomised controlled study.”, *BMJ: British Medical Journal*, nr 337, 2008, `http://www.bmj.com/content/bmj/337/bmj.a295.full.pdf`.

[2] `https://www.coachseye.com/`, Coach’s Eye, febr. 2015.

[3] `http://www.qualisys.com/applications/biomechanics/`, Qualisys Motion Capture Systems, febr. 2015.

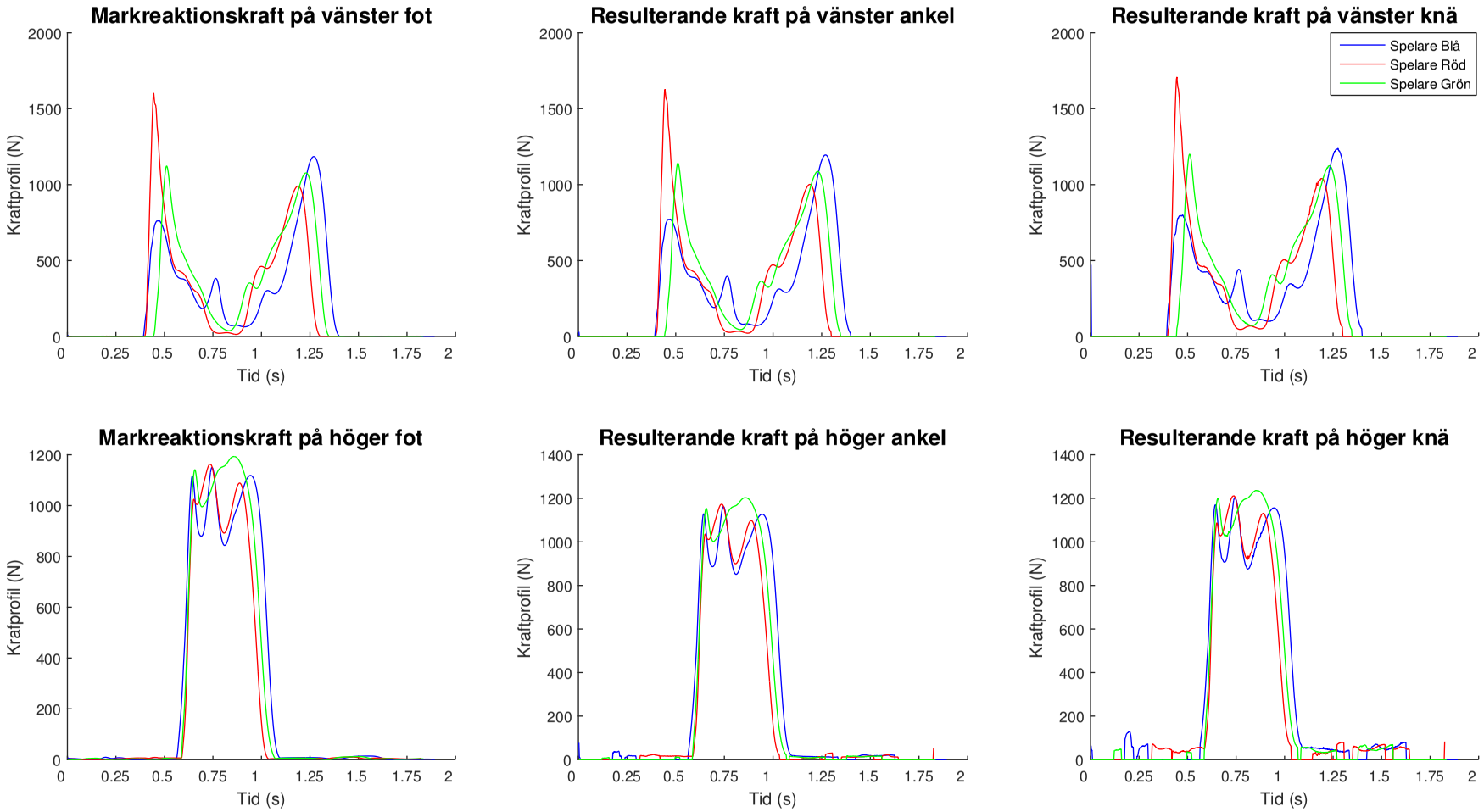
[4] D. Gordon, E. Robertson, G. Caldwell, J. Hamill, G. Kamen och S. Whittlesey, *Research Methods in Biomechanics*, 2:nd edition. Human Kinetics, 2004.

[5] J. L. Meriam och L. G. Kraige, *Engineering mechanics: Dynamics*. Hoboken, NJ: Wiley, 2012, vol. 2.

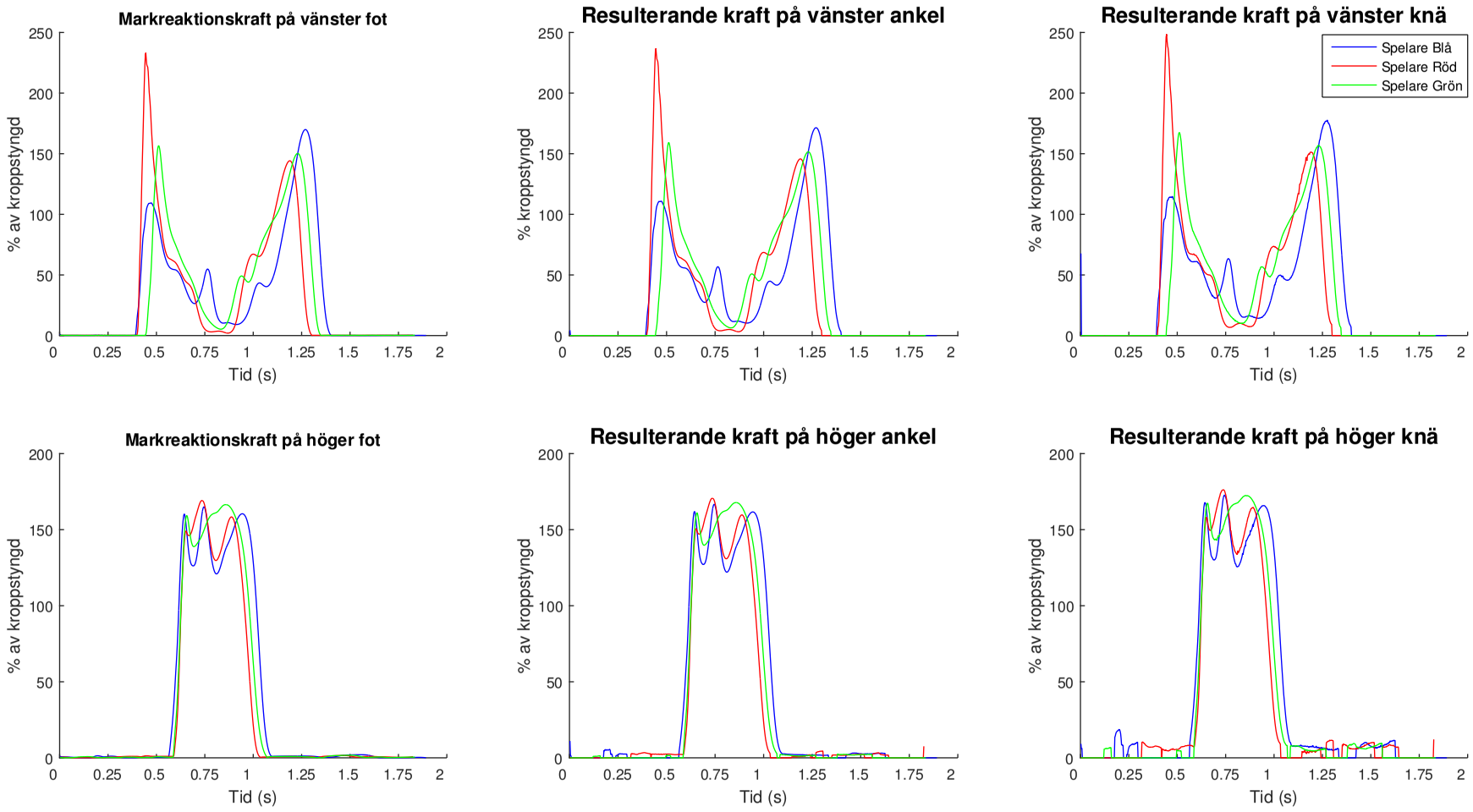
A

Grafer

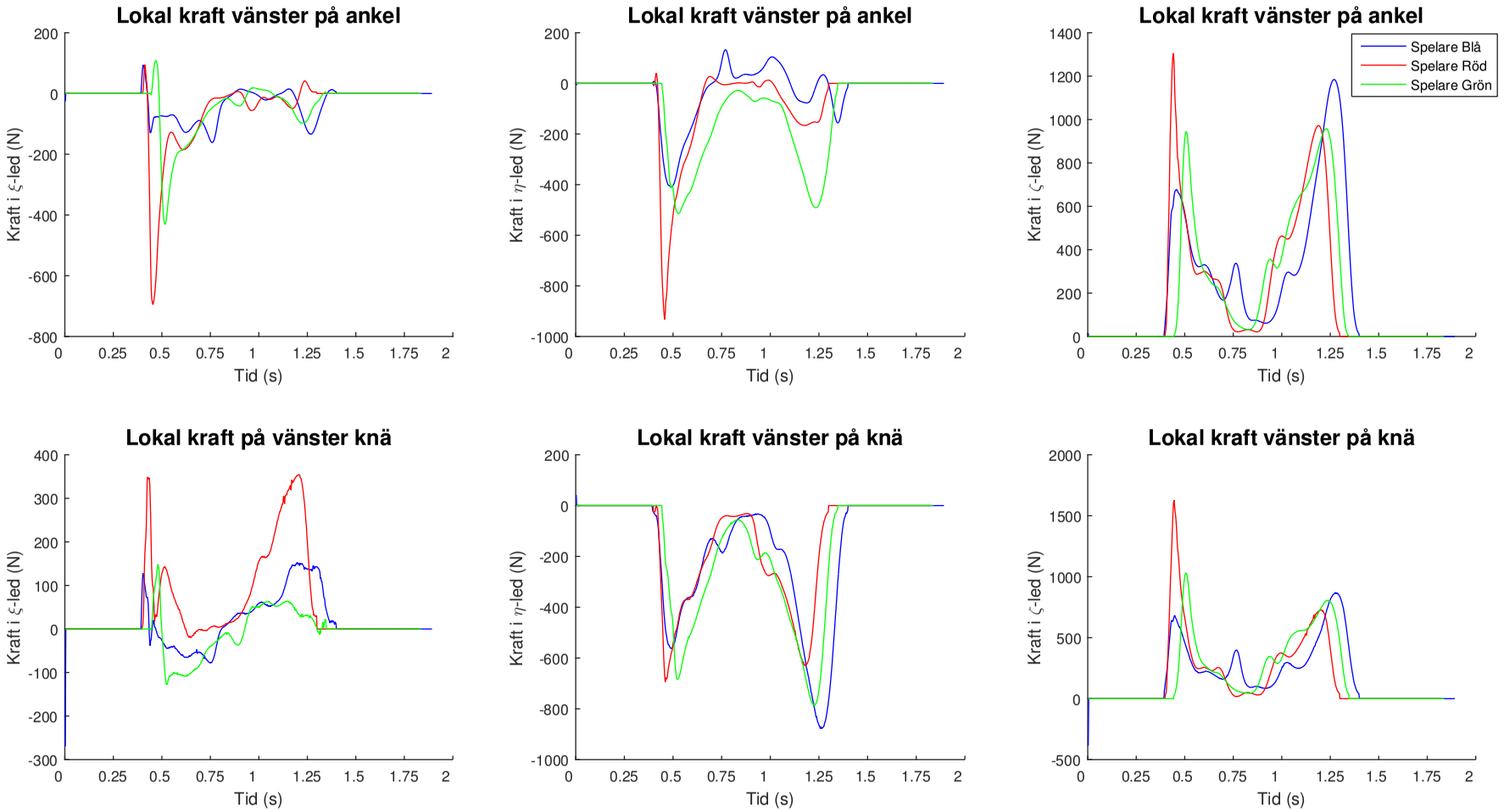
I detta appendix presenteras de grafer som resultaten bygger på. Alla graferna är genererade i MATLAB och de är färgkodade för att det ska bli lättare att hålla isär de olika spelarna.



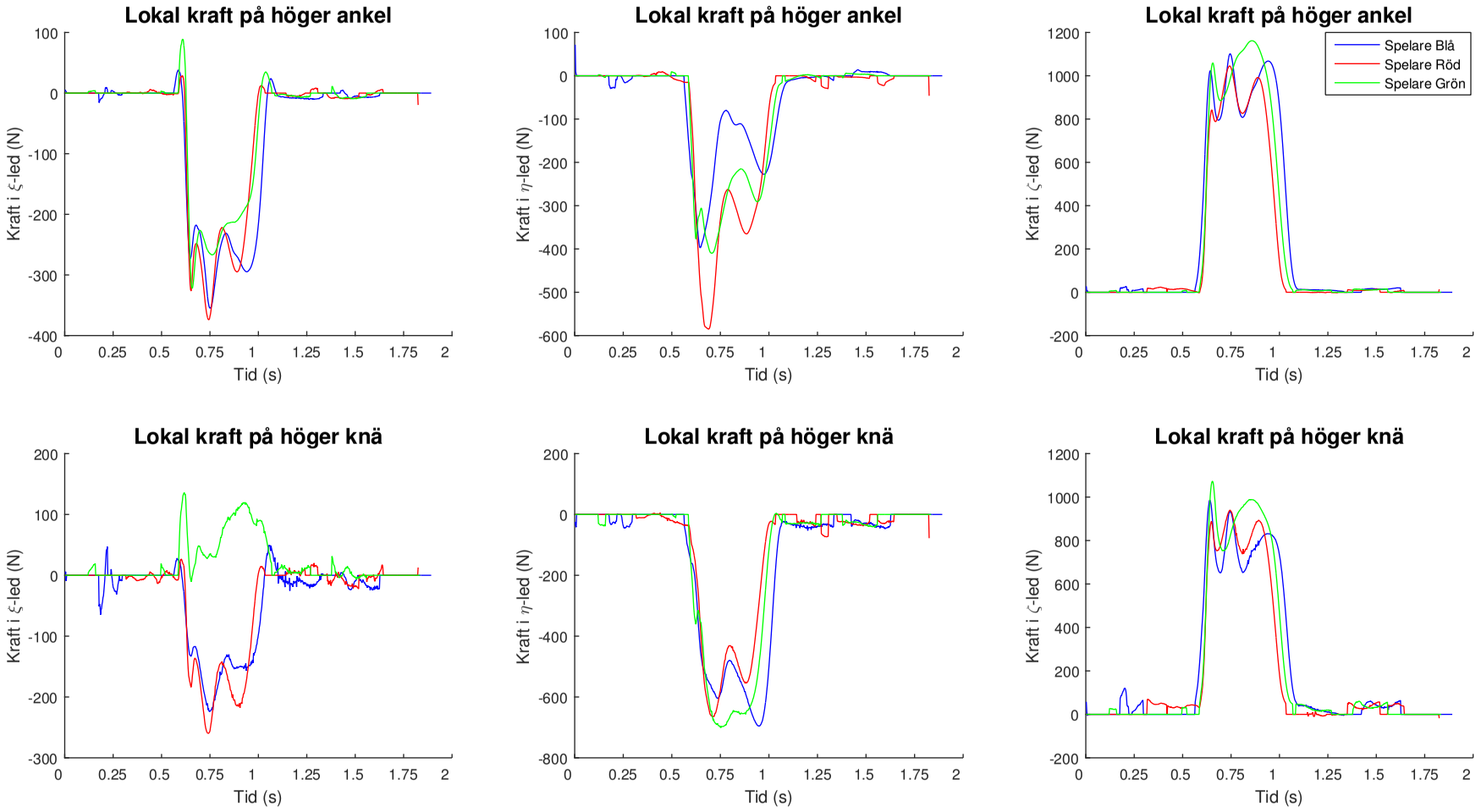
Figur A.1: Figuren visar storleken av den resulterande kraften på foten, ankeln och knät för respektive spelare som funktion av tid. Skalan på y-axeln är given i Newton. Notera att det förekommer brus innan och efter spelarna trampar på kraftplattorna.



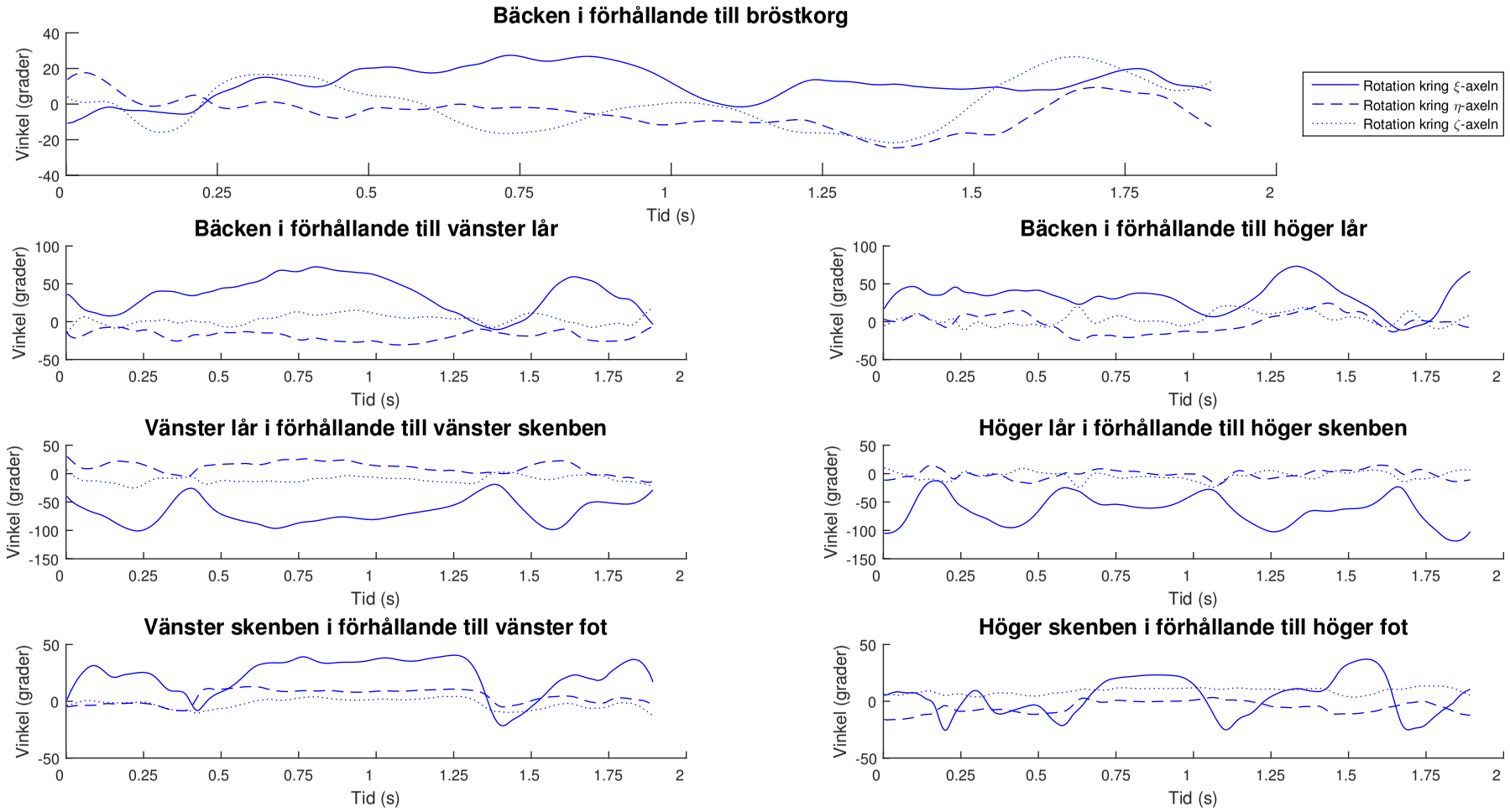
Figur A.2: Figuren visar storleken av den resulterande kraften på foten, ankeln och knät för respektive spelare som funktion av tid. För att underlätta en jämförelse mellan spelarna är y-axeln är skalad med % av kroppstyngden. Notera att det förekommer brus innan och efter spelarna trampar på kraftplattorna.



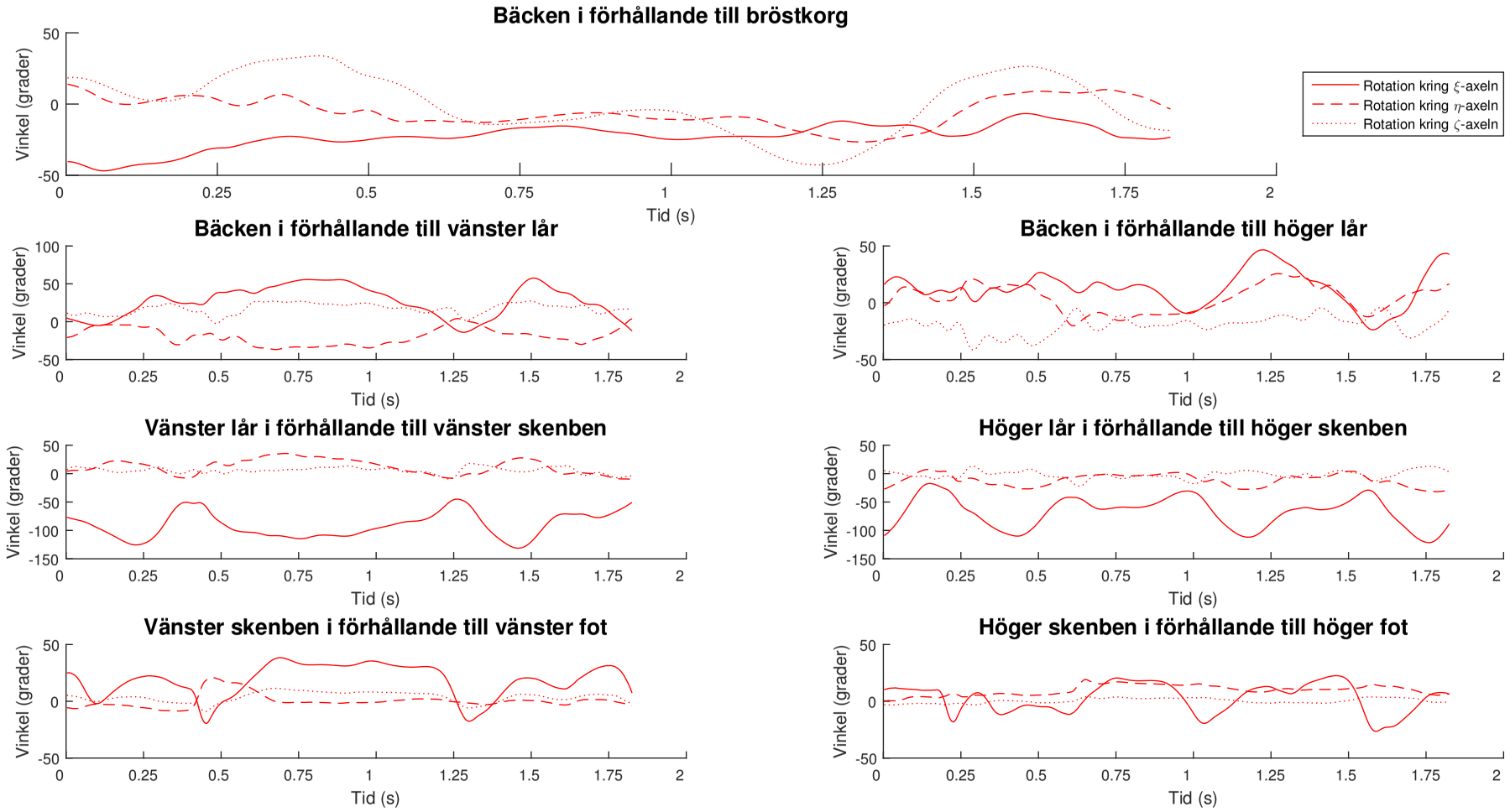
Figur A.3: Figuren illustrerar storleken av krafterna på vänster ankel respektive knä i spelarnas segmentcentrerade lokala koordinatsystem som funktion av tid. Enheten på y-axlen är här given i Newton.



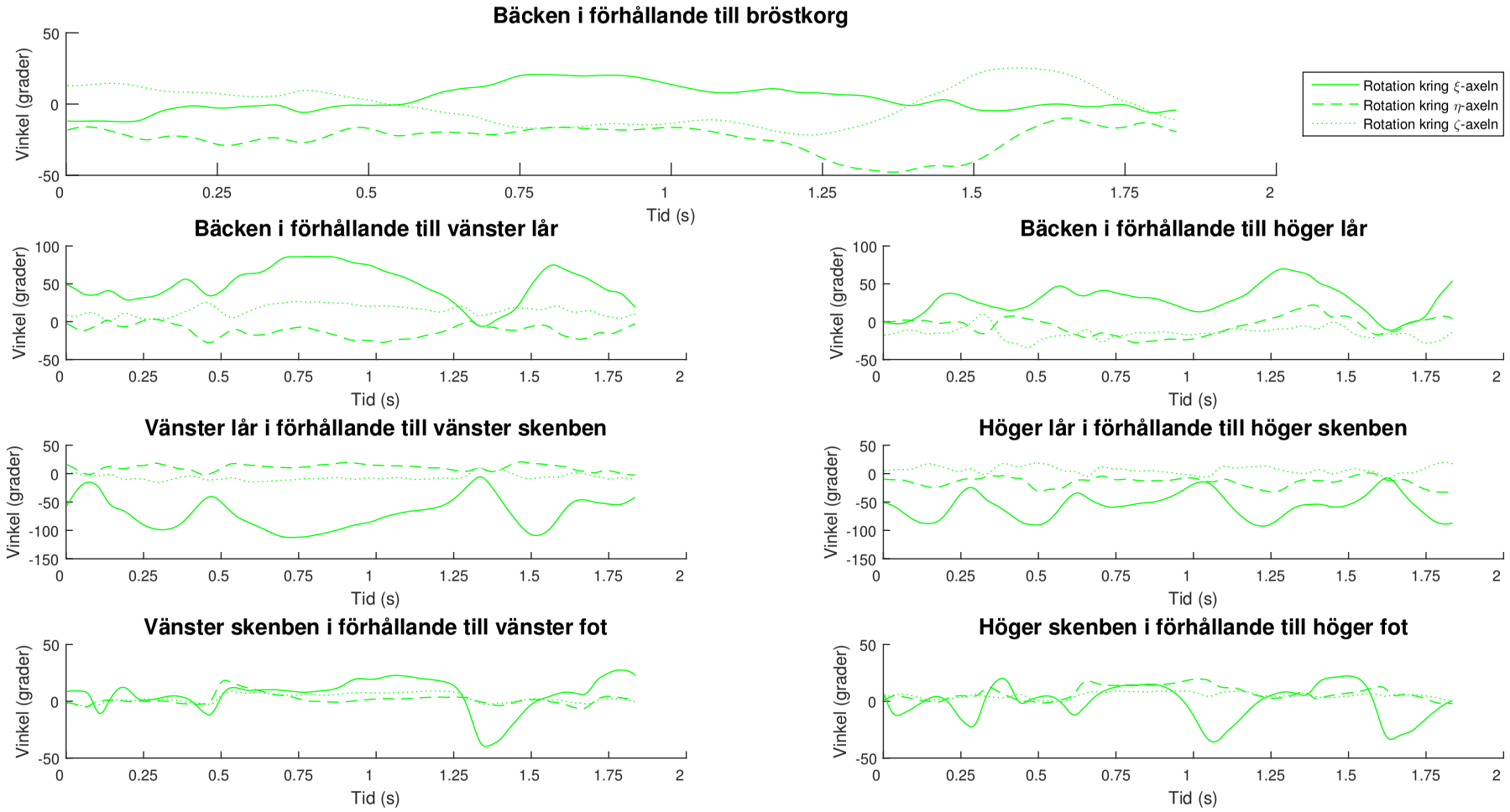
Figur A.4: Figuren illustrerar storleken av krafterna på höger ankel respektive knä i spelarnas segmentcentrerade lokala koordinatsystem som funktion av tid. Enheten på y-axlen är här given i Newton. Notera att det förekommer brus innan och efter spelarna trampar på kraftplattorna.



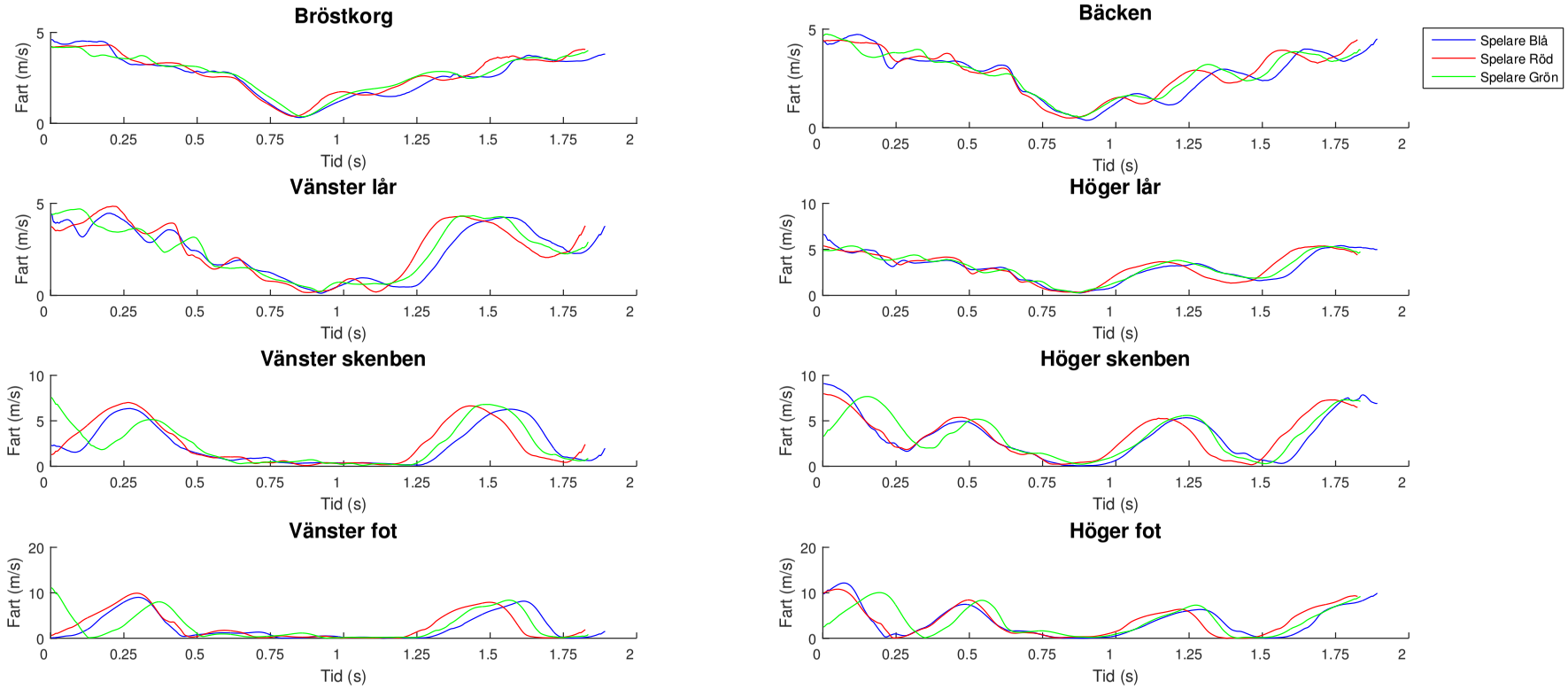
Figur A.5: Grafen visar hur segmenten för Spelare Blå vrider sig i förhållande till varandra under rörelsesekvensen, alltså hur det ena lokala koordinatsystemet roteras för ha samma orientering som det andra. Nollläget för dessa vinklar definieras från den statiska mätningen.



Figur A.6: Grafen visar hur segmenten för Spelare Röd vrider sig i förhållande till varandra under rörelsesekvensen, alltså hur det ena lokala koordinatsystemet roteras för ha samma orientering som det andra. Nollläget för dessa vinklar definieras från den statiska mätningen.



Figur A.7: Grafen visar hur segmenten för Spelare Grön vrider sig i förhållande till varandra under rörelsesekvensen, alltså hur det ena lokala koordinatsystemet roteras för ha samma orientering som det andra. Nollläget för dessa vinklar definieras från den statiska mätningen.



Figur A.8: Figuren visar farten för alla segment för respektive spelare som funktion av tid. Notera att det är fart och inte hastighet som visas eftersom rörelsesekvensen innefattar en vändning.

B

Matlab

I detta appendix presenteras den kod som skrivits i Matlab för att möjliggöra analysen av datan. Progammet följer den hierarkiska ordning beskriven i avsnitt 3.2 och kommer visualiseras på samma sätt här; `main` kommer först och sedan funktionsfilerna som denna anropar.

B.1 main

```
%%--------------------Import data to be evaluated--------------------------

clear all variables
close all
clc
static = ('path_to_file\static_data.mat');
S = importdata(static);
dynamic = ('path_to_file\dynamic_data.mat');
D = importdata(dynamic);
mass_of_person = 72;
labb = 'GU';
%% ------------------------- Player Blue ---------------------------------
clear all variables
close all
labb = 'Qualisys';
static = ('path_to_file\blue_player_static_data.mat');
S = importdata(static);
dynamic = ('path_to_file\blue_player_dynamic_data.mat');
D = importdata(dynamic);
mass_of_person = 71; % [kg]
%% ------------------------- Player Red ----------------------------------
clear all variables
close all
clc
labb = 'Qualisys';
static = ('path_to_file\red_player_static_data.mat');
S = importdata(static);
dynamic = ('path_to_file\red_player_dynamic_data.mat');
D = importdata(dynamic);
mass_of_person = 70; % [kg]
%% ------------------------- Player Green --------------------------------

clear all variables
close all
clc
labb = 'Qualisys';
static = ('path_to_file\green_player_static_data.mat');
S = importdata(static);
dynamic = ('path_to_file\green_player_dynamic_data.mat');
D = importdata(dynamic);
mass_of_person = 73; % [kg]

%% --------------------Define Header and Frames---------------------------

S_Header = S.Trajectories.Labeled.Labels;
S_Frames = S.Frames;
D_Header = D.Trajectories.Labeled.Labels;
D_Frames = D.Frames;

% -------------------Generating data in AllPointsMotion-------------------
[D_AllPointsMotion,D_index]=generate_data(D,'dynamic',labb);
```

```
[S_AllPointsMotion,S_index]=generate_data(S,'static',labb);


% ------------------Calculate static angles and segments-----------------
[static_angles,S_Segments] = static_evaluation(...
    S_AllPointsMotion,S_index);

% ------------------Calculate dynamic angles,index and segments----------
[dynamic_angles,D_Segments, index_segments, D_AllPointsMotion] = ...
    dynamic_evaluation( D_AllPointsMotion,static_angles,D_Frames,...
    D_index,labb);

% ------------------Calculate Mass---------------------------------------
[D_Segments]=calc_mass(D_AllPointsMotion,D_index,D_Frames,...
    D_Segments,index_segments,mass_of_person);

% ------------------Calculate velocity and acceleration------------------
D_Segments = velocity_acceleration(D_Frames,D_Segments,index_segments);

close all % if no plots
% ------------------Calculate Force--------------------------------------
[D_Segments]=force(D_AllPointsMotion,D_Segments,D_index,index_segments,...
    D_Frames,labb,mass_of_person,'None');

close all % if no plots

%% ------------------Plot all movements, dynamic-------------------------

% plot_movements(AllPointsMotion,Header, Frames,Segments,
% index_segments,labb,static_or_dynamic,lines_or_None,dots_or_None,
% LCS_or_None,force_or_None,All)

plot_movements(D_AllPointsMotion,D_Header,D_Frames,D_Segments,...
    index_segments,labb,'dynamic','lines','dots','None','force','None');

%% ------------------Plot all movements, static--------------------------
%plot_movements(AllPointsMotion,Header, Frames,Segments,
% index_segments,labb,static_or_dynamic,lines_or_None,dots_or_None,
% LCS_or_None,force_or_None,All)

plot_movements(S_AllPointsMotion,S_Header,S_Frames,S_Segments,...
    index_segments,labb,'static','lines','dots','LCS','None','None');

%% ------------------Plot joint angles from dynamic movements------------
clf
plot_dynamic_angles(D_Frames,dynamic_angles)
```

B.2 generate_data

```
function [AllPointsMotion, index]=generate_data(S, ...
    static_or_dynamic_as_a_string, lab)
% -------------------------------------------------------------------------
% Input  =  S : Structure containing data from a mat file.
%           static_or_dynamic_as_a_string : If generate_data shall
%                                           interpret the data as 'static'
%                                           or 'dynamic'.
%           lab : If the measurements were done at 'GU' or 'Qualisys' lab.
%
% Output =  AllPointsMotion : A structure which contains
%                             each markers position in PosTime and are now
%                             labeled and have a PlotRank.
%
%           index : A data structure that allows us to retrieve the
%                   index from a header using a corresponding key.
%
% The function compares each header with a bodypart which exist in the
% structure and generates its corresponding data.
% -------------------------------------------------------------------------

Header = S.Trajectories.Labeled.Labels;
index_names = {};
index_values = [];
% If force exists it needs to be smoothen.
if strcmp('dynamic',static_or_dynamic_as_a_string) && strcmp(lab,'GU')
    for i =1:length(S.Force)
        for j=1:3
    S.Force(i).COP(j,:) = smooth(S.Force(i).COP(j,:),45);
    S.Force(i).Moment(j,:) = smooth(S.Force(i).Moment(j,:),45);
    S.Force(i).Force(j,:) = smooth(S.Force(i).Force(j,:),45);
        end
    end
end
if strcmp('dynamic',static_or_dynamic_as_a_string) ...
        && strcmp(lab,'Qualisys')
    for i =1:length(S.Force)
        for j=1:3
    S.Force(i).COP(j,:) = smooth(S.Force(i).COP(j,:),15);
    S.Force(i).Moment(j,:) = smooth(S.Force(i).Moment(j,:),15);
    S.Force(i).Force(j,:) = smooth(S.Force(i).Force(j,:),15);
        end
    end
end

for i=1:length(Header)
    for j=1:4
        Data(i,j,:) = smooth(S.Trajectories.Labeled.Data(i,j,:),15);
    end
end

% Rearrange the matrix
```

```
Data = permute(Data,[3 2 1]);
% If static, a mean value of all position is requested
if strcmp('static',static_or_dynamic_as_a_string)
    for i = 1:S.Trajectories.Labeled.Count
        Data(1,:,i)=mean(Data(:,:,i));
    end
    Data = Data(1,:,:);
end
% generate_data is now going through each marker, naming it with its name,
% which body part (or place) it belongs to, its positions, its
% index (maps name with position) and a plotrank, which will later
% be used for plotting correct markers together and saves it into a
% structre, called AllPointsMotion. If the marker has forcedata, it saves
% it in AllPointsMotion aswell.

for i=1:S.Trajectories.Labeled.Count;
    AllPointsMotion(i).Header=Header{i};
    AllPointsMotion(i).PosTime= Data(:,1:3,i);

   if strcmp(AllPointsMotion(i).Header,'rtip');
        AllPointsMotion(i).Bodypart='right foot';
        AllPointsMotion(i).PlotRank = [1,7];
        index_names(i)={'rtip'};
        index_values(i)=i;
   elseif  strcmp(AllPointsMotion(i).Header,'rfm5')
        AllPointsMotion(i).Bodypart='right foot';
        AllPointsMotion(i).PlotRank = 2;
        index_names(i)={'rfm5'};
        index_values(i)=i;
   elseif strcmp(AllPointsMotion(i).Header,'rcalclat');
        AllPointsMotion(i).Bodypart='right foot';
        AllPointsMotion(i).PlotRank = 3;
        index_names(i)={'rcalclat'};
        index_values(i)=i;
   elseif  strcmp(AllPointsMotion(i).Header,'rheel');
        AllPointsMotion(i).Bodypart='right foot';
        AllPointsMotion(i).PlotRank = 4;
        index_names(i)={'rheel'};
        index_values(i)=i;
   elseif  strcmp(AllPointsMotion(i).Header,'rcalcmed');
        AllPointsMotion(i).Bodypart='right foot';
        AllPointsMotion(i).PlotRank = 5;
        index_names(i)={'rcalcmed'};
        index_values(i)=i;
   elseif strcmp(AllPointsMotion(i).Header,'rfm2');
        AllPointsMotion(i).Bodypart='right foot';
        AllPointsMotion(i).PlotRank = 6;
        index_names(i)={'rfm2'};
        index_values(i)=i;
   elseif strcmp(AllPointsMotion(i).Header,'ltip');
        AllPointsMotion(i).Bodypart='left foot';
        AllPointsMotion(i).PlotRank = [1,7];
        index_names(i)={'ltip'};
        index_values(i)=i;
   elseif  strcmp(AllPointsMotion(i).Header,'lfm5');
        AllPointsMotion(i).Bodypart='left foot';
```

```
        AllPointsMotion(i).PlotRank = 2;
        index_names(i)={'lfm5'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'lcalclat');
        AllPointsMotion(i).Bodypart='left foot';
        AllPointsMotion(i).PlotRank = 3;
        index_names(i)={'lcalclat'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'lheel');
        AllPointsMotion(i).Bodypart='left foot';
        AllPointsMotion(i).PlotRank = 4;
        index_names(i)={'lheel'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'lcalcmed');
        AllPointsMotion(i).Bodypart='left foot';
        AllPointsMotion(i).PlotRank = 5;
        index_names(i)={'lcalcmed'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'lfm2');
        AllPointsMotion(i).Bodypart='left foot';
        AllPointsMotion(i).PlotRank = 6;
        index_names(i)={'lfm2'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'lmalmed');
        AllPointsMotion(i).Bodypart = 'left shank';
        AllPointsMotion(i).PlotRank = 5;
        index_names(i)={'lmalmed'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'lmallat');
        AllPointsMotion(i).Bodypart = 'left shank';
        AllPointsMotion(i).PlotRank = 4;
        index_names(i)={'lmallat'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'ltibia');
        AllPointsMotion(i).Bodypart = 'left shank';
        AllPointsMotion(i).PlotRank = [3,6];
        index_names(i)={'ltibia'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'ltub');
        AllPointsMotion(i).Bodypart = 'left shank';
        AllPointsMotion(i).PlotRank = 2;
        index_names(i)={'ltub'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'ltiblat');
        AllPointsMotion(i).Bodypart = 'left shank';
        AllPointsMotion(i).PlotRank = 1;
        index_names(i)={'ltiblat'};
        index_values(i)=i;
     elseif  strcmp(AllPointsMotion(i).Header,'rmalmed');
        AllPointsMotion(i).Bodypart = 'right shank';
        AllPointsMotion(i).PlotRank = 5;
        index_names(i)={'rmalmed'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'rmallat');
        AllPointsMotion(i).Bodypart = 'right shank';
        AllPointsMotion(i).PlotRank = 4;
```

```
        index_names(i)={'rmallat'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'rtibia');
        AllPointsMotion(i).Bodypart = 'right shank';
        AllPointsMotion(i).PlotRank = [3,6];
        index_names(i)={'rtibia'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'rtub');
        AllPointsMotion(i).Bodypart = 'right shank';
        AllPointsMotion(i).PlotRank = 2;
        index_names(i)={'rtub'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'rtiblat');
        AllPointsMotion(i).Bodypart = 'right shank';
        AllPointsMotion(i).PlotRank = 1;
        index_names(i)={'rtiblat'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'lfel');
        AllPointsMotion(i).Bodypart='left thigh';
        AllPointsMotion(i).PlotRank = [1,4];
        index_names(i)={'lfel'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'lfem');
        AllPointsMotion(i).Bodypart='left thigh';
        AllPointsMotion(i).PlotRank = 2;
        index_names(i)={'lfem'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'ltroch');
        AllPointsMotion(i).Bodypart='left thigh';
        AllPointsMotion(i).PlotRank = 3;
        index_names(i)={'ltroch'};
        index_values(i)=i;
     elseif  strcmp(AllPointsMotion(i).Header,'rfel');
        AllPointsMotion(i).Bodypart='right thigh';
        AllPointsMotion(i).PlotRank = [1,4];
        index_names(i)={'rfel'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'rfem');
        AllPointsMotion(i).Bodypart='right thigh';
        AllPointsMotion(i).PlotRank = 2;
        index_names(i)={'rfem'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'rtroch');
        AllPointsMotion(i).Bodypart='right thigh';
        AllPointsMotion(i).PlotRank = 3;
        index_names(i)={'rtroch'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'rasis')
        AllPointsMotion(i).Bodypart='pelvis';
        AllPointsMotion(i).PlotRank = [1,5];
        index_names(i)={'rasis'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'lasis')
        AllPointsMotion(i).Bodypart='pelvis';
        AllPointsMotion(i).PlotRank = 2;
        index_names(i)={'lasis'};
```

```
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'lpsis')
        AllPointsMotion(i).Bodypart='pelvis';
        AllPointsMotion(i).PlotRank = 3;
        index_names(i)={'lpsis'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'rpsis')
        AllPointsMotion(i).Bodypart='pelvis';
        AllPointsMotion(i).PlotRank = 4;
        index_names(i)={'rpsis'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'sjn')
        AllPointsMotion(i).Bodypart='upper body';
        AllPointsMotion(i).PlotRank = [6,9];
        index_names(i)={'sjn'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'sxs')
        AllPointsMotion(i).Bodypart='upper body';
        AllPointsMotion(i).PlotRank = 10;
        index_names(i)={'sxs'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'rcaj')
        AllPointsMotion(i).Bodypart='upper body';
        AllPointsMotion(i).PlotRank = [3,7];
        index_names(i)={'rcaj'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'lcaj')
        AllPointsMotion(i).Bodypart='upper body';
        AllPointsMotion(i).PlotRank = [5,13];
        index_names(i)={'lcaj'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'c7_')
        AllPointsMotion(i).Bodypart='upper body';
        AllPointsMotion(i).PlotRank = [4,8,12];
        index_names(i)={'c7_'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'th8')
        AllPointsMotion(i).Bodypart='upper body';
        AllPointsMotion(i).PlotRank = 11;
        index_names(i)={'th8'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'rhand')
        AllPointsMotion(i).Bodypart='right arm';
        AllPointsMotion(i).PlotRank = 1;
        index_names(i)={'rhand'};
        index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'reb')
        AllPointsMotion(i).Bodypart ='right arm';
        AllPointsMotion(i).PlotRank = 2;
        index_names(i)={'reb'};
        index_values(i)=i;
    elseif  strcmp(AllPointsMotion(i).Header,'lhand')
        AllPointsMotion(i).Bodypart ='left arm';
        AllPointsMotion(i).PlotRank = 15;
        index_names(i)={'lhand'};
        index_values(i)=i;
```

```
    elseif strcmp(AllPointsMotion(i).Header,'leb')
         AllPointsMotion(i).Bodypart ='left arm';
         AllPointsMotion(i).PlotRank = [14,16];
         index_names(i)={'leb'};
         index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'Forceplate1') ...
            || strcmp(AllPointsMotion(i).Header,'fp1')
         AllPointsMotion(i).Bodypart = 'Global';
         AllPointsMotion(i).PlotRank = 1;
         index_names(i)={'fp1'};
         index_values(i)=i;
         AllPointsMotion(i).Force = - (S.Force(1).Force([2,1,3],:))';
         AllPointsMotion(i).COP = - (S.Force(1).COP([2,1,3],:))';
         AllPointsMotion(i).Torque_grf = - (S.Force(1).Moment([2,1,3],:))';
         AllPointsMotion(i).norm_Force = ...
             sqrt(sum(AllPointsMotion(i).Force.^2,2));

    elseif strcmp(AllPointsMotion(i).Header,'Forceplate4') ...
            || strcmp(AllPointsMotion(i).Header,'fp4')
         AllPointsMotion(i).Bodypart = 'Global';
         AllPointsMotion(i).PlotRank = 2;
         index_names(i)={'fp4'};
         index_values(i)=i;
         AllPointsMotion(i).Force = - (S.Force(2).Force([2,1,3],:))';
         AllPointsMotion(i).COP = - (S.Force(2).COP([2,1,3],:))';
         AllPointsMotion(i).Torque_grf = - (S.Force(2).Moment([2,1,3],:))';
         AllPointsMotion(i).norm_Force = ...
             sqrt(sum(AllPointsMotion(i).Force.^2,2));
    elseif strcmp(AllPointsMotion(i).Header,'Stick1')
         AllPointsMotion(i).Bodypart = 'Stick';
         AllPointsMotion(i).PlotRank = 1;
         index_names(i)={'stick1'};
         index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'Stick2')
         AllPointsMotion(i).Bodypart = 'Stick';
         AllPointsMotion(i).PlotRank = [2,4];
         index_names(i)={'stick2'};
         index_values(i)=i;
    elseif strcmp(AllPointsMotion(i).Header,'Stick3')
         AllPointsMotion(i).Bodypart = 'Stick';
         AllPointsMotion(i).PlotRank = 3;
         index_names(i) = {'stick3'};
         index_values(i) = i;
    end
end

% If the measurements were att Qualisys, fp1 and fp2 must be defined
% manually with their corresonding data.
if strcmp(static_or_dynamic_as_a_string,'dynamic') ...
         && strcmp(lab,'Qualisys')

     i = length(AllPointsMotion) + 1;
     AllPointsMotion(i).Header = 'fp1';
     AllPointsMotion(i).Bodypart = 'Global';
     index_names(i) = {'fp1'};
     index_values(i) = i;
```

```
    AllPointsMotion(i).Force = S.Force(1).Force';
    AllPointsMotion(i).COPglobal = S.Force(1).COP';
    AllPointsMotion(i).Torque_grf = S.Force(1).Moment';
    AllPointsMotion(i).norm_Force=sqrt(sum(AllPointsMotion(i).Force.^2,2));
    AllPointsMotion(i).PlotRank = 1;

    i = i + 1;
    AllPointsMotion(i).Header = 'fp2';
    AllPointsMotion(i).Bodypart = 'Global';
    index_names(i) = {'fp2'};
    index_values(i) = i;
    AllPointsMotion(i).Force = S.Force(2).Force';
    AllPointsMotion(i).COPglobal = S.Force(2).COP';
    AllPointsMotion(i).Torque_grf = S.Force(2).Moment';
    AllPointsMotion(i).norm_Force=sqrt(sum(AllPointsMotion(i).Force.^2,2));
    AllPointsMotion(i).PlotRank = 2;
end

% The index for all markers is created. containers.Map maps index_names
% with index_value.
index = containers.Map(index_names,index_values);


end
```

B.3 static_evaluation

```
function [static_angles, Segments] = static_evaluation(APM,index)
% ------------------------------------------------------------------------
% Input  = APM: Structure containing data from a mean value of the markers.
%          static_angles: static angles
%          D_Frames: Number of samples
%          index: index for every marker which and connected by
%                 their header.
%
% Output =  static_angles: Angles from static position
%           Segments: A data structure containing segment parts,
%                     their origins, and their corresponding data.
%                     Note that X,Y,Z isn't X-values, Y-values, Z-values.
%                     X contains the values (x,y,z) for the X-axis,
%                     Y contains the values (x,y,z) for the Y-axis and
%                     Z contains the values (x,y,z) for the Z-axis
%
% The function calculates the Rotation Matrix RM and sends it into the
% function joint_angles that returns the static angles of the segments.
% ------------------------------------------------------------------------

%Thorax segments LCS
%Origo for thorax

origin_thorax = APM(index('sjn')).PosTime;

% Normalvector to the plane
z_dir = 0.5 .* (APM(index('sjn')).PosTime+APM(index('c7_')).PosTime ...
    - (APM(index('sxs')).PosTime + APM(index('th8')).PosTime))...
    ./norm(0.5.*(APM(index('sjn')).PosTime+APM(index('c7_')).PosTime ...
    - (APM(index('sxs')).PosTime + APM(index('th8')).PosTime)));
x_dir=cross(APM(index('th8')).PosTime - APM(index('sxs')).PosTime,...
    APM(index('th8')).PosTime-APM(index('c7_')).PosTime)...
    ./norm(cross(abs(APM(index('th8')).PosTime ...
    - APM(index('sxs')).PosTime),abs(APM(index('th8')).PosTime ...
    - APM(index('c7_')).PosTime)));
y_dir = cross(z_dir,x_dir)./norm(cross(z_dir,x_dir));
% Scaling the axes
x_thorax = 150 .* x_dir+origin_thorax;
y_thorax = 150 .* y_dir+origin_thorax;
z_thorax = 150 .* z_dir+origin_thorax;

% Rotation matrix of the thorax
RM_thorax = [x_dir; y_dir; z_dir];

%Pelvis segment LCS
% Origo for the pelvis
origin_pelvis = 0.5 * (APM(index('rasis')).PosTime...
+ APM(index('lasis')).PosTime);
% Creating the x-component for the pelvis
x_dir=(APM(index('rasis')).PosTime - origin_pelvis)...
    ./norm(APM(index('rasis')).PosTime - origin_pelvis);
```

```
% Unit vector from the midpoint of rpsis and lpsis to origin
v_pelvis=(origin_pelvis-0.5.*(APM(index('rpsis')).PosTime ...
    + APM(index('lpsis')).PosTime)./norm(origin_pelvis - 0.5...
    .*(APM(index('rpsis')).PosTime+APM(index('lpsis')).PosTime));
% Normalvector to the plane
z_dir = cross(x_dir, v_pelvis);
y_dir = cross(z_dir, x_dir);

%Scaling the axes
x_pelvis = 150.*x_dir+origin_pelvis;
y_pelvis = 150.*y_dir+origin_pelvis;
z_pelvis = 150.*z_dir+origin_pelvis;
%Rotation matrix of the pelvis
RM_pelvis = [x_dir; y_dir; z_dir];

% Thigh segment LCS
% Left
origin_lthigh=0.5.*(APM(index('lfel')).PosTime+APM(index('lfem')).PosTime);
x_dir=(APM(index('lfem')).PosTime - origin_lthigh)...
    ./norm(APM(index('lfem')).PosTime - origin_lthigh);
v_lthigh=(APM(index('ltroch')).PosTime - origin_lthigh)...
    ./norm(APM(index('ltroch')).PosTime - origin_lthigh);
y_dir = cross(v_lthigh,x_dir);
z_dir = cross(x_dir, y_dir);
RM_lthigh = [x_dir; y_dir; z_dir];

% Scaling the axes
x_lthigh = 150.*x_dir + origin_lthigh;
y_lthigh = 150.*y_dir + origin_lthigh;
z_lthigh = 150.*z_dir + origin_lthigh;

% Right
origin_rthigh=0.5.*(APM(index('rfel')).PosTime+APM(index('rfem')).PosTime);
x_dir = (APM(index('rfel')).PosTime - origin_rthigh)...
    ./norm(APM(index('rfel')).PosTime - origin_rthigh);
v_rthigh=(APM(index('rtroch')).PosTime - origin_rthigh)...
    ./norm(APM(index('rtroch')).PosTime - origin_rthigh);
y_dir = cross(v_rthigh,x_dir);
z_dir = cross(x_dir,y_dir);
RM_rthigh = [x_dir; y_dir; z_dir];

x_rthigh = 150.*x_dir + origin_rthigh;
y_rthigh = 150.*y_dir + origin_rthigh;
z_rthigh = 150.*z_dir + origin_rthigh;

% Left shank segment LCS
origin_lshank = 0.5.*(APM(index('lmallat')).PosTime...
    + APM(index('lmalmed')).PosTime);
x_dir = (APM(index('lmalmed')).PosTime - origin_lshank)...
    ./norm(APM(index('lmalmed')).PosTime - origin_lshank);
v_lshank = (APM(index('ltub')).PosTime - origin_lshank)...
    ./norm(APM(index('ltub')).PosTime - origin_lshank);
y_dir = cross(v_lshank, x_dir);
z_dir = cross(x_dir, y_dir);
RM_lshank = [x_dir; y_dir; z_dir];
```

```
% Scaling the axes
x_lshank = 150.*x_dir + origin_lshank;
y_lshank = 150.*y_dir + origin_lshank;
z_lshank = 150.*z_dir + origin_lshank;

% Right shank segment LCS
origin_rshank = 0.5.*(APM(index('rmallat')).PosTime + ...
    APM(index('rmalmed')).PosTime);
x_dir = (APM(index('rmallat')).PosTime-origin_rshank)...
    ./norm(APM(index('rmallat')).PosTime - origin_rshank);
v_rshank = (APM(index('rtub')).PosTime - origin_rshank)...
    ./norm(APM(index('rtub')).PosTime - origin_rshank);
y_dir = cross(v_rshank, x_dir);
z_dir = cross(x_dir, y_dir);
RM_rshank = [x_dir; y_dir; z_dir];

% Scaling the axes
x_rshank = 150.*x_dir + origin_rshank;
y_rshank = 150.*y_dir + origin_rshank;
z_rshank = 150.*z_dir + origin_rshank;

% Foot segment LCS lcalcmed, lcalclat
% Left
origin_lfoot = 0.5.*(APM(index('lcalcmed')).PosTime...
    + APM(index('lcalclat')).PosTime);
x_dir = (APM(index('lcalcmed')).PosTime...
    - origin_lfoot)./norm(APM(index('lcalcmed')).PosTime - origin_lfoot);
v_lfoot = (origin_lfoot - APM(index('lheel')).PosTime)...
    ./norm(origin_lfoot - APM(index('lheel')).PosTime);
z_dir = cross(x_dir,v_lfoot);
y_dir = cross(z_dir, x_dir);
RM_lfoot = [x_dir; y_dir; z_dir];

% Scaling the axes
x_lfoot = 150.*x_dir + origin_lfoot;
y_lfoot = 150.*y_dir + origin_lfoot;
z_lfoot = 150.*z_dir + origin_lfoot;

% % Foot segment LCS rcalcmed, rcalclat
% Right
origin_rfoot = 0.5.*(APM(index('rcalcmed')).PosTime...
+ APM(index('rcalclat')).PosTime);
x_dir = (APM(index('rcalclat')).PosTime - origin_rfoot)...
    ./norm(APM(index('rcalclat')).PosTime-origin_rfoot);
v_rfoot = (origin_rfoot-APM(index('rheel')).PosTime)./norm(origin_rfoot ...
    - APM(index('rheel')).PosTime);
z_dir = cross(x_dir,v_rfoot);
y_dir = cross(z_dir, x_dir);
RM_rfoot = [x_dir; y_dir; z_dir];

% Scaling the axes
x_rfoot = 150.*x_dir+origin_rfoot;
y_rfoot = 150.*y_dir+origin_rfoot;
z_rfoot = 150.*z_dir+origin_rfoot;

% Computing the angles using RM matrixes
```

```
static_angles = zeros(7,3);
static_angles(1,:) = joint_angles(RM_pelvis,RM_thorax);
static_angles(2,:) = joint_angles(RM_pelvis,RM_lthigh);
static_angles(3,:) = joint_angles(RM_pelvis,RM_rthigh);
static_angles(4,:) = joint_angles(RM_lthigh,RM_lshank);
static_angles(5,:) = joint_angles(RM_rthigh,RM_rshank);
static_angles(6,:) = joint_angles(RM_lshank,RM_lfoot);
static_angles(7,:) = joint_angles(RM_rshank,RM_rfoot);

segments_parts = {'Left foot','Right foot','Left thigh','Right thigh',...
    'Left shank', 'Right shank', 'Thorax', 'Pelvis'};
segments_X = {x_lfoot,x_rfoot,x_lthigh,x_rthigh,x_lshank,...
    x_rshank, x_thorax,x_pelvis};
segments_Y ={y_lfoot,y_rfoot,y_lthigh,y_rthigh,y_lshank,...
    y_rshank, y_thorax,y_pelvis};
segments_Z ={z_lfoot,z_rfoot,z_lthigh,z_rthigh,z_lshank,...
    z_rshank, z_thorax,z_pelvis};
segments_origins = {origin_lfoot,origin_rfoot,origin_lthigh,...
    origin_rthigh,origin_lshank,origin_rshank,origin_thorax,origin_pelvis};

% Defines static segment structure
for i=1:length(segments_parts);
    Segments(i).Part = segments_parts(i);
    Segments(i).Origin = segments_origins{i};
    Segments(i).X = segments_X{i};
    Segments(i).Y = segments_Y{i};
    Segments(i).Z = segments_Z{i};
end
end
```

B.4 dynamic_evaluation

```
function [dynamic_angles,Segments,index_segments,APM] = dynamic_evaluation(...
    APM,static_angles,D_Frames,index,lab)
% -------------------------------------------------------------------------
% Input  = APM: Structure containing data from markers.
%          static_angles: static angles
%          D_Frames: Number of samples
%          index: index for every marker which and connected by
%                 their header.
%
% Output =  dynamic_angles: Angles from the dynamic movements
%           Segments: A data structure containing segment parts,
%                     their origins, and their corresponding data.
%                     Note that X,Y,Z isn't X-values, Y-values, Z-values.
%                     X contains the values (x,y,z) for the X-axis,
%                     Y contains the values (x,y,z) for the Y-axis and
%                     Z contains the values (x,y,z) for the Z-axis
%           index_segments: An index over all segments.
%           APM : AllPointsMotion, now containing Centre of Pressure's
%                 global coordinates.
%           lab : If the measurements were done at 'GU' or 'Qualisys' lab.
%                 If 'GU', Centre of Pressure global coordinates is
%                 calculated, otherwise not due to it already exists in APM
%
% The function calculates the Rotation Matrix RM and sends it into the
% function joint_angles which returns dynamic angles of segments.
% -------------------------------------------------------------------------


% Predefining a matrix for dynamic_angles for faster calculations
dynamic_angles = zeros(D_Frames,3,7);

for i = 1:D_Frames;

% Origo for thorax(i,:)
origin_thorax(i,:) = APM(index('sjn')).PosTime(i,:);

 % Normalvector x_dir, y_dir z_dir for Thorax to the plane and
 % its Origo origin_thorax:
z_dir = 0.5.*(APM(index('sjn')).PosTime(i,:)...
    + APM(index('c7_')).PosTime(i,:)-(APM(index('sxs')).PosTime(i,:)...
    + APM(index('th8')).PosTime(i,:)))...
    ./norm(0.5.*(APM(index('sjn')).PosTime(i,:)...
    + APM(index('c7_')).PosTime(i,:)-(APM(index('sxs')).PosTime(i,:)...
    + APM(index('th8')).PosTime(i,:))));

x_dir = cross(APM(index('th8')).PosTime(i,:)...
    - APM(index('c7_')).PosTime(i,:),APM(index('th8')).PosTime(i,:)...
    - APM(index('sxs')).PosTime(i,:))...
    ./ norm(cross(APM(index('th8')).PosTime(i,:)...
    - APM(index('sxs')).PosTime(i,:),APM(index('th8')).PosTime(i,:)...
    - APM(index('c7_')).PosTime(i,:)));
```

```
y_dir = cross(z_dir,x_dir)./norm(cross(z_dir,x_dir));

origin_thorax(i,:) = APM(index('sjn')).PosTime(i,:);

% Rotation matrix of the thorax
RM_thorax(:,:,i) = [x_dir; y_dir; z_dir];

% Scaling the axes
x_thorax(i,:) = 150.*x_dir+origin_thorax(i,:);
y_thorax(i,:) = 150.*y_dir+origin_thorax(i,:);
z_thorax(i,:) = 150.*z_dir+origin_thorax(i,:);

% Normalvector x_dir, y_dir z_dir for Pelvis to the plane
% and its Origo origin_pelvis, and a unit vector, v_pelvis,
%from the midpoint of rpsis(i,:) and lpsis(i,:) to origin:
origin_pelvis(i,:) = 0.5*(APM(index('rasis')).PosTime(i,:)...
    + APM(index('lasis')).PosTime(i,:));

x_dir=(APM(index('rasis')).PosTime(i,:)-origin_pelvis(i,:))...
    ./ norm(APM(index('rasis')).PosTime(i,:)-origin_pelvis(i,:));

v_pelvis(i,:)=(origin_pelvis(i,:)-0.5.*(APM(index('rpsis')).PosTime(i,:)...
    + APM(index('lpsis')).PosTime(i,:)))...
    ./ norm(origin_pelvis(i,:)-0.5.*(APM(index('rpsis')).PosTime(i,:)...
    + APM(index('lpsis')).PosTime(i,:)));
z_dir = cross(x_dir, v_pelvis(i,:));

y_dir=cross(z_dir, x_dir);
% Rotation matrix of the pelvis
RM_pelvis(:,:,i) = [x_dir; y_dir; z_dir];

%Scaling the axes
x_pelvis(i,:) = 150.*x_dir+origin_pelvis(i,:);
y_pelvis(i,:) = 150.*y_dir+origin_pelvis(i,:);
z_pelvis(i,:) = 150.*z_dir+origin_pelvis(i,:);

% Normalvector x_dir, y_dir z_dir for left thigh to the plane and its
% Origo origin_lthigh, and a unit vector, v_lthigh:
origin_lthigh(i,:)=0.5.*(APM(index('lfel')).PosTime(i,:)...
    +APM(index('lfem')).PosTime(i,:));
x_dir=(APM(index('lfem')).PosTime(i,:)-origin_lthigh(i,:))...
    ./norm(APM(index('lfem')).PosTime(i,:)-origin_lthigh(i,:));
v_lthigh(i,:)=(APM(index('ltroch')).PosTime(i,:)-origin_lthigh(i,:))...
    ./norm(APM(index('ltroch')).PosTime(i,:)-origin_lthigh(i,:));
y_dir=cross(v_lthigh(i,:),x_dir);
z_dir=cross(x_dir, y_dir);

% Rotation matrix of left thigh
RM_lthigh(:,:,i) = [x_dir; y_dir; z_dir];

% Scaling the axes
x_lthigh(i,:) = 150.*x_dir+origin_lthigh(i,:);
y_lthigh(i,:) = 150.*y_dir+origin_lthigh(i,:);
z_lthigh(i,:) = 150.*z_dir+origin_lthigh(i,:);
```

```
% Normalvector x_dir, y_dir z_dir for right thigh to the plane
% and its Origo origin_rthigh,  and a unit vector, v_rthigh:
origin_rthigh(i,:)=0.5.*(APM(index('rfel')).PosTime(i,:)...
    +APM(index('rfem')).PosTime(i,:));
x_dir=(APM(index('rfel')).PosTime(i,:)-origin_rthigh(i,:))...
    ./norm(APM(index('rfel')).PosTime(i,:)-origin_rthigh(i,:));
v_rthigh(i,:)=(APM(index('rtroch')).PosTime(i,:)-origin_rthigh(i,:))...
    ./norm(APM(index('rtroch')).PosTime(i,:)-origin_rthigh(i,:));
y_dir=cross(v_rthigh(i,:),x_dir);
z_dir=cross(x_dir,y_dir);
% Rotation matrix of right thigh
RM_rthigh(:,:,i) = [x_dir; y_dir; z_dir];

% Scaling the axes
x_rthigh(i,:) = 150.*x_dir+origin_rthigh(i,:);
y_rthigh(i,:) = 150.*y_dir+origin_rthigh(i,:);
z_rthigh(i,:) = 150.*z_dir+origin_rthigh(i,:);

% Normalvector x_dir, y_dir z_dir for left shank to the plane
% and its Origo origin_lshank,  and a unit vector, v_lshank:
origin_lshank(i,:)=0.5.*(APM(index('lmallat')).PosTime(i,:)...
    +APM(index('lmalmed')).PosTime(i,:));
x_dir=(APM(index('lmalmed')).PosTime(i,:)-origin_lshank(i,:))...
    ./norm(APM(index('lmalmed')).PosTime(i,:)-origin_lshank(i,:));
v_lshank(i,:)=(APM(index('ltub')).PosTime(i,:)-origin_lshank(i,:))...
    ./norm(APM(index('ltub')).PosTime(i,:)-origin_lshank(i,:));
y_dir=cross(v_lshank(i,:), x_dir);
z_dir=cross(x_dir, y_dir);
% Rotation matrix of left shank
RM_lshank(:,:,i) = [x_dir; y_dir; z_dir];

% Scaling the axes
x_lshank(i,:) = 150.*x_dir+origin_lshank(i,:);
y_lshank(i,:) = 150.*y_dir+origin_lshank(i,:);
z_lshank(i,:) = 150.*z_dir+origin_lshank(i,:);

% Normalvector x_dir, y_dir z_dir for right shank to the plane and its
% Origo origin_rshank, and a unit vector, v_rshank:
origin_rshank(i,:)=0.5.*(APM(index('rmallat')).PosTime(i,:)...
    +APM(index('rmalmed')).PosTime(i,:));
x_dir=(APM(index('rmallat')).PosTime(i,:)-origin_rshank(i,:))...
    ./norm(APM(index('rmallat')).PosTime(i,:)-origin_rshank(i,:));
v_rshank(i,:)=(APM(index('rtub')).PosTime(i,:)-origin_rshank(i,:))...
    ./norm(APM(index('rtub')).PosTime(i,:)-origin_rshank(i,:));
y_dir=cross(v_rshank(i,:), x_dir);
z_dir=cross(x_dir, y_dir);
% Rotation matrix of right shank
RM_rshank(:,:,i) = [x_dir; y_dir; z_dir];

% Scaling the axes
x_rshank(i,:) = 150.*x_dir+origin_rshank(i,:);
y_rshank(i,:) = 150.*y_dir+origin_rshank(i,:);
z_rshank(i,:) = 150.*z_dir+origin_rshank(i,:);

% Normalvector x_dir, y_dir z_dir for left foot to the plane and its
% Origo origin_lfoot, and a unit vector, v_lfoot:
```

```
origin_lfoot(i,:)=0.5.*(APM(index('lcalcmed')).PosTime(i,:)...
    +APM(index('lcalclat')).PosTime(i,:));
x_dir=(APM(index('lcalcmed')).PosTime(i,:)-origin_lfoot(i,:))...
    ./norm(APM(index('lcalcmed')).PosTime(i,:)-origin_lfoot(i,:));
v_lfoot(i,:)=(origin_lfoot(i,:)-APM(index('lheel')).PosTime(i,:))...
    ./norm(origin_lfoot(i,:)-APM(index('lheel')).PosTime(i,:));
z_dir=cross(x_dir,v_lfoot(i,:));
y_dir=cross(z_dir, x_dir);
% Rotation matrix of left foot
RM_lfoot(:,:,i) = [x_dir; y_dir; z_dir];

% Scaling the axes
x_lfoot(i,:) = 150.*x_dir+origin_lfoot(i,:);
y_lfoot(i,:) = 150.*y_dir+origin_lfoot(i,:);
z_lfoot(i,:) = 150.*z_dir+origin_lfoot(i,:);

% Normalvector x_dir, y_dir z_dir for right foot to the plane and its
% Origo origin_rfoot, and a unit vector, v_rfoot:
origin_rfoot(i,:)=0.5.*(APM(index('rcalcmed')).PosTime(i,:)...
    + APM(index('rcalclat')).PosTime(i,:));
x_dir = (APM(index('rcalclat')).PosTime(i,:) - origin_rfoot(i,:))...
    ./ norm(APM(index('rcalclat')).PosTime(i,:) - origin_rfoot(i,:));
v_rfoot(i,:) = (origin_rfoot(i,:) - APM(index('rheel')).PosTime(i,:))...
    ./ norm(origin_rfoot(i,:) - APM(index('rheel')).PosTime(i,:));
z_dir = cross(x_dir,v_rfoot(i,:));
y_dir = cross(z_dir, x_dir);
% Rotation matrix of right foot
RM_rfoot(:,:,i) = [x_dir; y_dir; z_dir];

% Scaling the axes
x_rfoot(i,:) = 150.*x_dir + origin_rfoot(i,:);
y_rfoot(i,:) = 150.*y_dir + origin_rfoot(i,:);
z_rfoot(i,:) = 150.*z_dir + origin_rfoot(i,:);

% The dynamic angles are calculated by subtracting joint_angles from
% static_angles. The joint angles are calculated by inserting the
% RM matrices in the function  joint_angles.

dynamic_angles(i,:,1) = static_angles(1,:)...
    - joint_angles(RM_pelvis(:,:,i),RM_thorax(:,:,i));
dynamic_angles(i,:,2) = static_angles(2,:)...
    - joint_angles(RM_pelvis(:,:,i),RM_lthigh(:,:,i));
dynamic_angles(i,:,3) = static_angles(3,:)...
    - joint_angles(RM_pelvis(:,:,i),RM_rthigh(:,:,i));
dynamic_angles(i,:,4) = static_angles(4,:)...
    - joint_angles(RM_lthigh(:,:,i),RM_lshank(:,:,i));
dynamic_angles(i,:,5) = static_angles(5,:)...
    - joint_angles(RM_rthigh(:,:,i),RM_rshank(:,:,i));
dynamic_angles(i,:,6) = static_angles(6,:)...
    - joint_angles(RM_lshank(:,:,i),RM_lfoot(:,:,i));
dynamic_angles(i,:,7) = static_angles(7,:)...
    - joint_angles(RM_rshank(:,:,i),RM_rfoot(:,:,i));

end
for j = 1:7
    for i = 1:D_Frames-1
```

```
        if abs(dynamic_angles(i,1,j)-dynamic_angles(i+1,1,j)) > 30
            if dynamic_angles(i+1,1,j) > 0
                dynamic_angles(i+1,1,j) = dynamic_angles(i+1,1,j)-180;
            else
                dynamic_angles(i+1,1,j) = dynamic_angles(i+1,1,j)+180;
            end
        end
    end
end

% Calculating Centre Of Pressure
if strcmp(lab,'GU')
    % Marker at force plate to local-origin force plate
    d = [350 210 0];
    % The forceplate samples 3 times faster than QMT.
    % Therefore, j is created to only take every third value
    % in Centre Of Pressure.
    j= 1;
    for i =1:length(APM(index('fp1')).PosTime)
        APM(index('fp1')).COPglobal(i,:) =APM(index('fp1')).PosTime(i,:)...
            + d + APM(index('fp1')).COP(j,:);
        APM(index('fp4')).COPglobal(i,:) =APM(index('fp4')).PosTime(i,:)...
            + d + APM(index('fp4')).COP(j,:);
        j= j+3;
     end
end

% For later calculations and plots, a structure, Segments, is created
% containing segment parts, their origins and their corresponding data.
% Note that X,Y,Z isn't X-values, Y-values, Z-values. X contains the values
% (x,y,z) for the X-axis, Y contains the values (x,y,z) for the Y-axis and
% Z contains the values (x,y,z) for the Z-axis

segments_parts = {'lfoot','rfoot','lthigh','rthigh','lshank',...
    'rshank', 'thorax', 'pelvis'};
segments_X = {x_lfoot,x_rfoot,x_lthigh,x_rthigh,x_lshank,....
    x_rshank, x_thorax,x_pelvis};
segments_Y ={y_lfoot,y_rfoot,y_lthigh,y_rthigh,y_lshank,...
    y_rshank, y_thorax,y_pelvis};
segments_Z ={z_lfoot,z_rfoot,z_lthigh,z_rthigh,z_lshank,...
    z_rshank, z_thorax,z_pelvis};
segments_origins = {origin_lfoot,origin_rfoot,origin_lthigh,...
    origin_rthigh,origin_lshank,origin_rshank,origin_thorax,origin_pelvis};
segments_RM = {RM_lfoot,RM_rfoot,RM_lthigh,RM_rthigh,RM_lshank,...
    RM_rshank,RM_thorax,RM_pelvis};

% Creating an index for Segments
index_number=zeros(1,length(segments_parts));
for i = 1:length(segments_parts)
    index_number(i) = i;
end
% An index is created to facilitate later calculations for Segments.
index_segments = containers.Map(segments_parts,index_number);

% Defining Segments and their fields
for i=1:length(segments_parts);
```

```
    Segments(i).Part = segments_parts(i);
    Segments(i).Origin = segments_origins{i};
    Segments(i).X = segments_X{i};
    Segments(i).Y = segments_Y{i};
    Segments(i).Z = segments_Z{i};
    Segments(i).RM = segments_RM{i};
end
end
```

B.5 joint_angles

```
function [ angle_vector ] = joint_angles( RM_matrix_1,RM_matrix_2 )
% -------------------------------------------------------------------------
% Input  = RM_matrix_1: Rotation matrix 1
%          RM_matrix_2: Rotation matrix 2
%
% Output = angle_vector: vector with the three angles:
%                        alpha (rotation around x-axis)
%                        beta (rotation around y-axis)
%                        gamma (rotation around z-axis)
%
% Function that calculates the angles between segments using equations
% directly derived from the decomposition matrix. For more detailed
% information on the decomposition matrix and equations, see
% "Research Methods In BIOMECHANICS" second edition by D.Gordon et al.
% page 52, equations  2.64, 2.65, 2.66, 2.67.
% -------------------------------------------------------------------------

R = RM_matrix_1*RM_matrix_2';

alpha = atand(-R(3,2)/R(3,3));
beta = atand(R(3,1)/sqrt(R(1,1)^2+R(2,1)^2));
gamma = atand(-R(2,1)/R(1,1));

angle_vector = [alpha, beta, gamma];

end
```

B.6 calc_mass

```
function [Segments] = calc_mass(APM,index,Frames,Segments,...
    index_segments, mass_of_person)
% -------------------------------------------------------------------------
% Input  =  APM : Structure containing all markers motion.
%           index : index for every marker connected by
%                   their header.
%           Segments : A data structure containing segment parts and
%                      corresponding data.
%           Frames: Number of samples.
%           index_segments: index for every segment connected by
%                           their header.
%           mass_of_person: mass of the person in the test (in kg)
%
%
% Output = Segments: A data structure containing segment parts,
%                    including their mass.
%
% Caluclates segment mass by Dempster's Body Segment Parameters
% which approximates a percentage of each segment's mass and calculates
% Centre Of Mass, COM, for each segment.
% -------------------------------------------------------------------------

Segments(index_segments('lfoot')).Mass = 0.0145*mass_of_person;
Segments(index_segments('rfoot')).Mass = 0.0145*mass_of_person;
Segments(index_segments('lshank')).Mass = 0.0465*mass_of_person;
Segments(index_segments('rshank')).Mass = 0.0465*mass_of_person;
Segments(index_segments('lthigh')).Mass = 0.1*mass_of_person;
Segments(index_segments('rthigh')).Mass = 0.1*mass_of_person;
Segments(index_segments('thorax')).Mass = 0.216*mass_of_person;
Segments(index_segments('pelvis')).Mass = 0.142*mass_of_person;

for i=1:Frames
% Feet
Segments(index_segments('lfoot')).COM(i,:) = ...
    APM(index('lheel')).PosTime(i,:)-(APM(index('lheel')).PosTime(i,:)...
    - APM(index('ltip')).PosTime(i,:)).*0.429;
Segments(index_segments('rfoot')).COM(i,:) = ...
    APM(index('rheel')).PosTime(i,:)-(APM(index('rheel')).PosTime(i,:)...
    - APM(index('rtip')).PosTime(i,:)).*0.429;
%Shanks
Segments(index_segments('lshank')).COM(i,:) =...
    APM(index('ltub')).PosTime(i,:)-(APM(index('ltub')).PosTime(i,:)...
    - Segments(index_segments('lshank')).Origin(i,:)).*0.433;
Segments(index_segments('rshank')).COM(i,:) = ...
    APM(index('rtub')).PosTime(i,:) - (APM(index('rtub')).PosTime(i,:)...
    - Segments(index_segments('rshank')).Origin(i,:)).*0.433;
%Thighs
Segments(index_segments('lthigh')).COM(i,:) = ...
    APM(index('ltroch')).PosTime(i,:)-(APM(index('ltroch')).PosTime(i,:)...
    - Segments(index_segments('lthigh')).Origin(i,:)).*0.433;
Segments(index_segments('rthigh')).COM(i,:) = ...
```

```
    APM(index('rtroch')).PosTime(i,:)-(APM(index('rtroch')).PosTime(i,:)...
    - Segments(index_segments('rthigh')).Origin(i,:)).*0.433;
%Pelvis
Segments(index_segments('pelvis')).COM(i,:) =...
    (APM(index('rpsis')).PosTime(i,:)...
    + APM(index('lpsis')).PosTime(i,:)).*0.25 ...
    + (APM(index('rasis')).PosTime(i,:) ...
    + APM(index('lasis')).PosTime(i,:)).*0.25;
%Thorax
Segments(index_segments('thorax')).COM(i,:) = ...
    (APM(index('th8')).PosTime(i,:) + APM(index('sxs')).PosTime(i,:)).*0.5;

end
```

B.7 velocity_acceleration

```
function [ D_Segments ] = velocity_acceleration(D_Frames,D_Segments,...
    index_segments)
% -------------------------------------------------------------------------
% Input  = AllPointsMotion: Structure containing all markers motion
%          D_Frames: Number of samples.
%          D_Segments: A data structure containing dynamic
%          segmentparts and their data
%          index_segments: index for every segment which are connected by
%                          their header.
%
% Output = D_Segments, also containing speed,velocity and accelertion
%
% The function calculates each segments speed, velocity and acceleration
% by numerical differentiation (forward, backwards and central).
% The function also plotts each segments speed and acceleration.
% -------------------------------------------------------------------------

% Windowing Planck-taper
% To remove unstable values at the beginning and at the end that
% arises due to the smooth-function in generate_data.
w = tukeywin(D_Frames, 0.15);

for i=1:length(D_Segments)


%Forward velocity
D_Segments(i).speed(1) = norm(D_Segments(i).COM(2,:)...
    - D_Segments(i).COM(1,:))/(1000/400);
D_Segments(i).velocity(1,:) = (D_Segments(i).COM(2,:)...
    - D_Segments(i).COM(1,:))./(1000/400);
%Backward velocity
D_Segments(i).speed(length(D_Segments(i).COM)) = ...
    norm(D_Segments(i).COM(length(D_Segments(i).COM),:)...
    - D_Segments(i).COM(length(D_Segments(i).COM)-1,:))/(1000/400);
D_Segments(i).velocity(length(D_Segments(i).COM),:) = ...
    (D_Segments(i).COM(length(D_Segments(i).COM),:)...
    - D_Segments(i).COM(length(D_Segments(i).COM)-1,:))./(1000/400);

%Forward acceleration
D_Segments(i).acceleration(1,:) = (D_Segments(i).COM(1,:)...
    - 2*D_Segments(i).COM(2,:)+D_Segments(i).COM(3,:))/(1/400)^2/1000;
D_Segments(i).abs_acceleration(1) = norm(D_Segments(i).acceleration(1,:));

%Backward acceleration
D_Segments(i).acceleration(length(D_Segments(i).COM),:) =...
    (D_Segments(i).COM(length(D_Segments(i).COM),:)...
    - 2*D_Segments(i).COM(length(D_Segments(i).COM)-1,:)...
    + D_Segments(i).COM(length(D_Segments(i).COM)-2,:))/(1/400)^2/1000;
D_Segments(i).abs_acceleration(length(D_Segments(i).COM)) =...
    norm(D_Segments(i).acceleration(length(D_Segments(i).COM),:));
```

```
%Central velocity and acceleration
    for j = 2:length(D_Segments(i).COM)-1
        D_Segments(i).speed(j) = norm(D_Segments(i).COM(j+1,:)...
            - D_Segments(i).COM(j-1,:))/(2000/400);
        D_Segments(i).velocity(j,:) = (D_Segments(i).COM(j+1,:)...
            - D_Segments(i).COM(j-1,:))./(2000/400);
        D_Segments(i).acceleration(j,:) = (D_Segments(i).COM(j+1,:)...
            - 2*D_Segments(i).COM(j,:) + D_Segments(i).COM(j-1,:))...
            /(2/400)^2/1000;
        D_Segments(i).abs_acceleration(j) =...
            norm(D_Segments(i).acceleration(j,:));
    end
end

for i=1:length(D_Segments)
    D_Segments(i).abs_acceleration = w'.*D_Segments(i).abs_acceleration;
end

% Time scaling
up = ceil(D_Frames*4)/4;
down = floor(D_Frames*4)/4;

if D_Frames - down < up - D_Frames;
    timestop = down/400;
else
    timestop = up/400;
end
v = 0;
j =1;
for i = 1:0.25:timestop+1;

    v(end+1) = v(j)+0.25;
    j =j+1;
end
clf
figure(1)
set(gcf, 'name', 'Hastighet')

subplot(4,2,1)
hold on
title('thorax')
plot(1:D_Frames,D_Segments(index_segments('thorax')).speed(1,:))

xlabel('Tid (s)')
ylabel('Hastighet (m/s)')
set(gca,'XTickLabel',v)

subplot(4,2,2)
hold on
title('pelvis')
plot(1:D_Frames,D_Segments(index_segments('pelvis')).speed(1,:))

xlabel('Tid (s)')
ylabel('Hastighet (m/s)')
set(gca,'XTickLabel',v)
```

```
subplot(4,2,3)
hold on
title('lthigh')
plot(1:D_Frames,D_Segments(index_segments('lthigh')).speed(1,:))

xlabel('Tid (s)')
ylabel('Hastighet (m/s)')
set(gca,'XTickLabel',v)

subplot(4,2,4)
hold on
title('rthigh')
plot(1:D_Frames,D_Segments(index_segments('rthigh')).speed(1,:))

xlabel('Tid (s)')
ylabel('Hastighet (m/s)')
set(gca,'XTickLabel',v)

subplot(4,2,5)
hold on
title('lshank')
plot(1:D_Frames,D_Segments(index_segments('lshank')).speed(1,:))

xlabel('Tid (s)')
ylabel('Hastighet (m/s)')
set(gca,'XTickLabel',v)

subplot(4,2,6)
hold on
title('rshank')
plot(1:D_Frames,D_Segments(index_segments('rshank')).speed(1,:))

xlabel('Tid (s)')
ylabel('Hastighet (m/s)')
set(gca,'XTickLabel',v)

subplot(4,2,7)
hold on
title('lfoot')
plot(1:D_Frames,D_Segments(index_segments('lfoot')).speed(1,:))

xlabel('Tid (s)')
ylabel('Hastighet (m/s)')
set(gca,'XTickLabel',v)

subplot(4,2,8)
hold on
title('rfoot')
plot(1:D_Frames,D_Segments(index_segments('rfoot')).speed(1,:))

xlabel('Tid (s)')
ylabel('Hastighet (m/s)')
set(gca,'XTickLabel',v)

figure(2)
set(gcf, 'name', 'Acceleration')
```

```
subplot(4,2,1)
hold on
title('thorax')
plot(1:D_Frames,D_Segments(index_segments('thorax')).abs_acceleration)

xlabel('Tid (s)')
ylabel('Acceleration (m/s^2)')
set(gca,'XTickLabel',v)

subplot(4,2,2)
hold on
title('pelvis')
plot(1:D_Frames,D_Segments(index_segments('pelvis')).abs_acceleration)

xlabel('Tid (s)')
ylabel('Acceleration (m/s^2)')
set(gca,'XTickLabel',v)

subplot(4,2,3)
hold on
title('lthigh')
plot(1:D_Frames,D_Segments(index_segments('lthigh')).abs_acceleration)

xlabel('Tid (s)')
ylabel('Acceleration (m/s^2)')
set(gca,'XTickLabel',v)

subplot(4,2,4)
hold on
title('rthigh')
plot(1:D_Frames,D_Segments(index_segments('rthigh')).abs_acceleration)

xlabel('Tid (s)')
ylabel('Acceleration (m/s^2)')
set(gca,'XTickLabel',v)

subplot(4,2,5)
hold on
title('lshank')
plot(1:D_Frames,D_Segments(index_segments('lshank')).abs_acceleration)

xlabel('Tid (s)')
ylabel('Acceleration (m/s^2)')
set(gca,'XTickLabel',v)

subplot(4,2,6)
hold on
title('rshank')
plot(1:D_Frames,D_Segments(index_segments('rshank')).abs_acceleration)

xlabel('Tid (s)')
ylabel('Acceleration (m/s^2)')
set(gca,'XTickLabel',v)

subplot(4,2,7)
```

```
hold on
title('lfoot')
plot(1:D_Frames,D_Segments(index_segments('lfoot')).abs_acceleration)

xlabel('Tid (s)')
ylabel('Acceleration (m/s^2)')
set(gca,'XTickLabel',v)

subplot(4,2,8)
hold on
title('rfoot')
plot(1:D_Frames,D_Segments(index_segments('rfoot')).abs_acceleration)

xlabel('Tid (s)')
ylabel('Acceleration (m/s^2)')
set(gca,'XTickLabel',v)
end
```

B.8 force

```
function [ Segments ] = force( APM,Segments,index,index_seg,...
    D_Frames,lab,mp,impulse_or_None)
% -------------------------------------------------------------------------
% Input  =  APM: AllPointsMotion. A structure containing all markers
%               motion.
%           Segments: A data structure containing dynamic segmentparts
%           and corresponding data
%           index: Index for all markers
%           index_segments: Index for all Segment parts
%           D_Frames: Number of samples
%           mp : mass_of_person
%           lab : If the measurements were done at 'GU' or 'Qualisys' lab.
%           impulse_or_None : Set active ('impusle')to calculate impulse.
%
% Output = Segments: also containing force data
%
% The function calculates forces on Segment parts using Newton's second law
% of motion F = ma.
% -------------------------------------------------------------------------

clf
% Gravity
g = [0 0 -9.82];
% The forceplate samples 3 times faster than QMT. Therefore, j is created
% to only take every third value in Centre Of Pressure.
if strcmp(lab,'GU')
    j= 1;
    for i =1:length(APM(index('fp1')).PosTime)
    distance_left(i,:) = (APM(index('fp1')).PosTime(i,:)...
        - Segments(index_seg('lfoot')).COM(i,:));
    distance_right(i,:) = (APM(index('fp1')).PosTime(i,:)...
        - Segments(index_seg('rfoot')).COM(i,:));
    j= j+3;
    end

    % The conditions checks which foot is closest to the two different
    % forceplates. When it is determined, the function calculates
    % forces on each segment.
    if min(sqrt(sum(distance_left.^2,2)))...
            < min(sqrt(sum(distance_right.^2,2)))
        j=1;
        for i=1:D_Frames
            if (APM(index('fp1')).norm_Force(j,:)) > 5
                Segments(index_seg('lfoot')).force(i,:) = ...
                    Segments(index_seg('lfoot')).Mass ...
                    .*(Segments(index_seg('lfoot')).acceleration(i,:)-g)...
                    - APM(index('fp1')).Force(j,:);
                Segments(index_seg('lshank')).force(i,:) = ...
                    Segments(index_seg('lshank')).Mass.*...
                    (Segments(index_seg('lshank')).acceleration(i,:) -g)...
                    + Segments(index_seg('lfoot')).force(i,:);
```

```
        else
            Segments(index_seg('lfoot')).force(i,:) = [0 0 0];
            Segments(index_seg('lshank')).force(i,:) = [0 0 0];
        end

        if (APM(index('fp4')).norm_Force(j,:)) > 5
            Segments(index_seg('rfoot')).force(i,:) = ...
                Segments(index_seg('rfoot')).Mass...
                .*(Segments(index_seg('rfoot')).acceleration(i,:)-g)...
                -(APM(index('fp4')).Force(j,:));
            Segments(index_seg('rshank')).force(i,:) =...
                Segments(index_seg('rshank')).Mass.* ...
                (Segments(index_seg('rshank')).acceleration(i,:)- g)...
                + Segments(index_seg('rfoot')).force(i,:);
        else
            Segments(index_seg('rfoot')).force(i,:) = [0 0 0];
            Segments(index_seg('rshank')).force(i,:) = [0 0 0];
        end
        j=j+3;
    end
elseif min(sqrt(sum(distance_left.^2,2)))...
        > min(sqrt(sum(distance_right.^2,2)))
    j=1;
    for i=1:D_Frames
        if (APM(index('fp4')).norm_Force(j,:)) > 5
            Segments(index_seg('lfoot')).force(i,:) = ...
                Segments(index_seg('lfoot')).Mass.*...
                (Segments(index_seg('lfoot')).acceleration(i,:) - g)...
                - (APM(index('fp4')).Force(j,:));
            Segments(index_seg('lshank')).force(i,:) = ...
                Segments(index_seg('lshank')).Mass.* ...
                (Segments(index_seg('lshank')).acceleration(i,:)- g)...
                + Segments(index_seg('lfoot')).force(i,:);
        else
            Segments(index_seg('lfoot')).force(i,:) = [0 0 0];
            Segments(index_seg('lshank')).force(i,:) = [0 0 0];
        end

        if (APM(index('fp1')).norm_Force(j,:)) > 5
            Segments(index_seg('rfoot')).force(i,:) = ...
                Segments(index_seg('rfoot')).Mass...
                .*(Segments(index_seg('rfoot')).acceleration(i,:)-g)...
                - (APM(index('fp1')).Force(j,:));
            Segments(index_seg('rshank')).force(i,:) =...
                Segments(index_seg('rshank')).Mass...
            .*(Segments(index_seg('rshank')).acceleration(i,:) - g)...
            + Segments(index_seg('rfoot')).force(i,:);
        else
            Segments(index_seg('rfoot')).force(i,:) = [0 0 0];
            Segments(index_seg('rshank')).force(i,:) = [0 0 0];
        end
        j=j+3;
    end
else
    msg = 'Error occurred - Bad data';
    error(msg)
```

```
    end
elseif strcmp(lab,'Qualisys')
    if max(APM(index('ltip')).PosTime(:,1))...
            < max(APM(index('rtip')).PosTime(:,1))
        for i=1:D_Frames;
            if (APM(index('fp2')).norm_Force(i,:)) > 5
                Segments(index_seg('lfoot')).force(i,:) = ...
                    Segments(index_seg('lfoot')).Mass ...
                .*(Segments(index_seg('lfoot')).acceleration(i,:) - g)...
                - APM(index('fp2')).Force(i,:);
                Segments(index_seg('lshank')).force(i,:) = ...
                    Segments(index_seg('lshank')).Mass...
                .*(Segments(index_seg('lshank')).acceleration(i,:) - g)...
                + Segments(index_seg('lfoot')).force(i,:);
            else
                Segments(index_seg('lfoot')).force(i,:) = [0 0 0];
                Segments(index_seg('lshank')).force(i,:) = [0 0 0];
            end

            if (APM(index('fp1')).norm_Force(i,:)) > 5
                Segments(index_seg('rfoot')).force(i,:) = ...
                    Segments(index_seg('rfoot')).Mass...
                    .*(Segments(index_seg('rfoot')).acceleration(i,:)-g)...
                    - (APM(index('fp1')).Force(i,:));
                Segments(index_seg('rshank')).force(i,:) =...
                    Segments(index_seg('rshank')).Mass.*...
                    (Segments(index_seg('rshank')).acceleration(i,:)- g)...
                    + Segments(index_seg('rfoot')).force(i,:);
            else
                Segments(index_seg('rfoot')).force(i,:) = [0 0 0];
                Segments(index_seg('rshank')).force(i,:) = [0 0 0];
            end
        end
    elseif max(APM(index('ltip')).PosTime(:,1))...
            > max(APM(index('rtip')).PosTime(:,1))

        for i=1:D_Frames;
            if (APM(index('fp1')).norm_Force(i,:)) > 5
                Segments(index_seg('lfoot')).force(i,:) = ...
                    Segments(index_seg('lfoot')).Mass...
                    .*(Segments(index_seg('lfoot')).acceleration(i,:)-g)...
                    - APM(index('fp1')).Force(i,:);
                Segments(index_seg('lshank')).force(i,:) = ...
                    Segments(index_seg('lshank')).Mass ...
                .*(Segments(index_seg('lshank')).acceleration(i,:) - g)...
                + Segments(index_seg('lfoot')).force(i,:);
            else
                Segments(index_seg('lfoot')).force(i,:) = [0 0 0];
                Segments(index_seg('lshank')).force(i,:) = [0 0 0];
            end

            if (APM(index('fp2')).norm_Force(i,:)) > 5
                Segments(index_seg('rfoot')).force(i,:) = ...
                    Segments(index_seg('rfoot')).Mass...
                    .*(Segments(index_seg('rfoot')).acceleration(i,:)-g)...
                    - (APM(index('fp2')).Force(i,:));
```

```
                Segments(index_seg('rshank')).force(i,:) = ...
                    Segments(index_seg('rshank')).Mass...
                .*(Segments(index_seg('rshank')).acceleration(i,:) - g)...
                + Segments(index_seg('rfoot')).force(i,:);
            else
                Segments(index_seg('rfoot')).force(i,:) = [0 0 0];
                Segments(index_seg('rshank')).force(i,:) = [0 0 0];
            end
        end
    end
end

% Euclidean norm of the force
Segments(index_seg('lfoot')).norm_force = ...
    sqrt(sum(Segments(index_seg('lfoot')).force.^2,2));
Segments(index_seg('rfoot')).norm_force = ...
    sqrt(sum(Segments(index_seg('rfoot')).force.^2,2));
Segments(index_seg('lshank')).norm_force = ...
    sqrt(sum(Segments(index_seg('lshank')).force.^2,2));
Segments(index_seg('rshank')).norm_force = ...
    sqrt(sum(Segments(index_seg('rshank')).force.^2,2));
% Point of attack from force
Segments(index_seg('lfoot')).Point = (APM(index('lmalmed')).PosTime...
    + APM(index('lmallat')).PosTime)/2;
Segments(index_seg('rfoot')).Point = (APM(index('rmalmed')).PosTime...
    + APM(index('rmallat')).PosTime)/2;
Segments(index_seg('lshank')).Point = APM(index('ltub')).PosTime;
Segments(index_seg('rshank')).Point = APM(index('rtub')).PosTime;

for i = 1:D_Frames
   Segments(index_seg('rfoot')).local_force(:,i) = ...
       Segments(index_seg('rfoot')).RM(:,:,i) ...
       * Segments(index_seg('rfoot')).force(i,:)';
   Segments(index_seg('lfoot')).local_force(:,i) = ...
       Segments(index_seg('lfoot')).RM(:,:,i)...
       * Segments(index_seg('lfoot')).force(i,:)';
   Segments(index_seg('rshank')).local_force(:,i) = ...
       Segments(index_seg('rshank')).RM(:,:,i) ...
       * Segments(index_seg('rshank')).force(i,:)';
   Segments(index_seg('lshank')).local_force(:,i) = ...
       Segments(index_seg('lshank')).RM(:,:,i) ...
       *Segments(index_seg('lshank')).force(i,:)';
end

% Time scaling
up = ceil(D_Frames*4)/4;
down = floor(D_Frames*4)/4;

if D_Frames - down < up - D_Frames;
    timestop = down/400;
else
    timestop = up/400;
end
v = [0];
j =1;
for i = 1:0.25:timestop+1;
```

```
    v(end+1) = v(j)+0.25;
    j =j+1;
end
if strcmp(lab,'GU')
    forceplate = 'fp4';
else
    forceplate = 'fp2';
end

% Plots
figure(1)
set(gcf, 'name', 'Kraft')
subplot(2,3,1)
hold on
title('V\"anster fot', 'Interpreter', 'LaTex')
if strcmp('Qualisys',lab)
    plot(1:D_Frames,APM(index(forceplate)).norm_Force)
else
    plot(1:D_Frames,APM(index(forceplate)).norm_Force...
        (1:3:length(APM(index(forceplate)).norm_Force)))
end

xlabel('Tid (s)')
ylabel('Markreaktionskraft (N)')
set(gca,'XTickLabel',v)
subplot(2,3,4)
hold on
title('H\"oger fot', 'Interpreter', 'LaTex')
% GU samples 3 times faster than Qualisys.
if strcmp('Qualisys',lab)
    plot(1:D_Frames,APM(index('fp1')).norm_Force)
else
    plot(1:D_Frames,APM(index('fp1')).norm_Force...
        (1:3:length(APM(index('fp1')).norm_Force)))
end

xlabel('Tid (s)')
ylabel('Markreaktionskraft (N)')
set(gca,'XTickLabel',v)

subplot(2,3,2)
hold on
title('V\"anster ankel', 'Interpreter', 'LaTex')
plot(1:D_Frames,Segments(index_seg('lfoot')).norm_force)
xlabel('Tid (s)')
ylabel('Kraft p\aa{} ankel (N)', 'Interpreter', 'LaTex')
set(gca,'XTickLabel',v)
subplot(2,3,5)
hold on
title('H\"oger ankel', 'Interpreter', 'LaTex')
plot(1:D_Frames,Segments(index_seg('rfoot')).norm_force)
xlabel('Tid (s)')
ylabel('Kraft p\aa{} ankel (N)')
set(gca,'XTickLabel',v)
```

```
subplot(2,3,3)
hold on
title('V\"anster kn\"a', 'Interpreter', 'LaTex')
plot(1:D_Frames,Segments(index_seg('lshank')).norm_force)
xlabel('Tid (s)')
ylabel('Kraft p\aa{} kn\"a (N)', 'Interpreter', 'LaTex')
set(gca,'XTickLabel',v)
subplot(2,3,6)
hold on
title('H\"oger kn\"a', 'Interpreter', 'LaTex')
plot(1:D_Frames,Segments(index_seg('rshank')).norm_force)
xlabel('Tid (s)')
ylabel('Kraft p\aa{} kn\"a (N)', 'Interpreter', 'LaTex')
set(gca,'XTickLabel',v)

% Using ginput to manually set the limits of x1 to x2 in plots in order to
% calculate the impulse.
if strcmp('impulse',impulse_or_None)
[x,y] = ginput(2);
% Dividing with sampleframes to get [N*s] instead of [N*frames]
I1 = trapz(APM(index(forceplate)).norm_Force(ceil(x(1)):ceil(x(2)),1))/400

[x,y] = ginput(2);
I2 = trapz(APM(index('fp1')).norm_Force(ceil(x(1)):ceil(x(2)),1))/400

[x,y] = ginput(2);
I3 = trapz(APM(index(forceplate)).norm_Force(ceil(x(1)):ceil(x(2)),1))/400
end
figure(2)
set(gcf, 'name', '% kroppstyngd')
subplot(2,3,2)
hold on
title('Resulterande kraft p\aa{} v\"anster ankel', 'Interpreter', 'LaTex')
plot(1:D_Frames,Segments(index_seg('lfoot')).norm_force/(mp*9.82)*100)
xlabel('Tid (s)')
ylabel('% kroppstyngd')
set(gca,'XTickLabel',v)
subplot(2,3,5)
hold on
title('Resulterande kraft p\aa{} h\"oger ankel', 'Interpreter', 'LaTex')
plot(1:D_Frames,Segments(index_seg('rfoot')).norm_force/(mp*9.82)*100)
xlabel('Tid (s)')
ylabel('% kroppstyngd')
set(gca,'XTickLabel',v)

subplot(2,3,3)
hold on
title('Resulterande kraft p\aa{} v\"anster kn\"a', 'Interpreter', 'LaTex')
plot(1:D_Frames,Segments(index_seg('lshank')).norm_force/(mp*9.82)*100)
xlabel('Tid (s)')
ylabel('% kroppstyngd')
set(gca,'XTickLabel',v)
subplot(2,3,6)
hold on
title('Resulterande kraft p\aa{} h\"oger kn\"a', 'Interpreter', 'LaTex')
plot(1:D_Frames,Segments(index_seg('rshank')).norm_force/(mp*9.82)*100)
```

```
xlabel('Tid (s)')
ylabel('% kroppstyngd')
set(gca,'XTickLabel',v)

subplot(2,3,1)
hold on
title('Markreaktionskraft f\"or v\"anster fot', 'Interpreter', 'LaTex')
if strcmp('Qualisys',lab)
    plot(1:D_Frames,APM(index(forceplate)).norm_Force/(mp*9.82)*100)
else
    plot(1:D_Frames,APM(index(forceplate)).norm_Force...
        (1:3:length(APM(index(forceplate)).norm_Force),:)/(mp*9.82)*100)
end
xlabel('Tid (s)')
ylabel('% kroppstyngd')
set(gca,'XTickLabel',v)
subplot(2,3,4)
hold on
title('Markreaktionskraft f\"or h\"oger fot', 'Interpreter', 'LaTex')
if strcmp('Qualisys',lab)
    plot(1:D_Frames,APM(index('fp1')).norm_Force/(mp*9.82)*100)
else
    plot(1:D_Frames,APM(index('fp1')).norm_Force...
        (1:3:length(APM(index('fp1')).norm_Force))/(mp*9.82)*100)
end
xlabel('Tid (s)')
ylabel('% kroppstyngd')
set(gca,'XTickLabel',v)

figure(3)
set(gcf, 'name', 'Lokala krafter')
subplot(2,3,1)
hold on
title('Lokal kraft h\"oger ankel', 'Interpreter', 'LaTex')
plot(1:D_Frames, Segments(index_seg('rfoot')).local_force(1,:))
xlabel('Tid (s)')
ylabel('Kraft i \xi-led (N)')
set(gca,'XTickLabel',v)
subplot(2,3,2)
hold on
title('Lokal kraft h\"oger ankel', 'Interpreter', 'LaTex')
plot(1:D_Frames, Segments(index_seg('rfoot')).local_force(2,:))
xlabel('Tid (s)')
ylabel('Kraft i \eta-led (N)')
set(gca,'XTickLabel',v)
subplot(2,3,3)
hold on
title('Lokal kraft h\"oger ankel', 'Interpreter', 'LaTex')
plot(1:D_Frames, Segments(index_seg('rfoot')).local_force(3,:))
xlabel('Tid (s)')
ylabel('Kraft i \zeta-led (N)')
set(gca,'XTickLabel',v)
subplot(2,3,4)
hold on
title('Lokal kraft h\"oger kn\"a', 'Interpreter', 'LaTex')
plot(1:D_Frames, Segments(index_seg('rshank')).local_force(1,:))
```

```
xlabel('Tid (s)')
ylabel('Kraft i \xi-led (N)')
set(gca,'XTickLabel',v)
subplot(2,3,5)
hold on
title('Lokal kraft h\"oger kn\"a', 'Interpreter', 'LaTex')
plot(1:D_Frames, Segments(index_seg('rshank')).local_force(2,:))
xlabel('Tid (s)')
ylabel('Kraft i \eta-led (N)')
set(gca,'XTickLabel',v)
subplot(2,3,6)
hold on
title('Lokal kraft h\"oger kn"a', 'Interpreter', 'LaTex')
plot(1:D_Frames, Segments(index_seg('rshank')).local_force(3,:))
xlabel('Tid (s)')
ylabel('Kraft i \zeta-led (N)')
set(gca,'XTickLabel',v)

figure(4)
set(gcf, 'name', 'Lokala krafter')
subplot(2,3,1)
hold on
title('Lokal kraft v\"anster ankel', 'Interpreter', 'LaTex')
plot(1:D_Frames, Segments(index_seg('lfoot')).local_force(1,:))
xlabel('Tid (s)')
ylabel('Kraft i \xi-led (N)')
set(gca,'XTickLabel',v)
subplot(2,3,2)
hold on
title('Lokal kraft v\"anster ankel', 'Interpreter', 'LaTex')
plot(1:D_Frames, Segments(index_seg('lfoot')).local_force(2,:))
xlabel('Tid (s)')
ylabel('Kraft i \eta-led (N)')
set(gca,'XTickLabel',v)
subplot(2,3,3)
hold on
title('Lokal kraft v\"anster ankel', 'Interpreter', 'LaTex')
plot(1:D_Frames, Segments(index_seg('lfoot')).local_force(3,:))
xlabel('Tid (s)')
ylabel('Kraft i \zeta-led (N)')
set(gca,'XTickLabel',v)

subplot(2,3,4)
hold on
title('Lokal kraft v\"anster kn\"a', 'Interpreter', 'LaTex')
plot(1:D_Frames, Segments(index_seg('lshank')).local_force(1,:))
xlabel('Tid (s)')
ylabel('Kraft i \xi-led (N)')
set(gca,'XTickLabel',v)
subplot(2,3,5)
hold on
title('Lokal kraft v\"anster kn\"a', 'Interpreter', 'LaTex')
plot(1:D_Frames, Segments(index_seg('lshank')).local_force(2,:))
xlabel('Tid (s)')
ylabel('Kraft i \eta-led (N)')
set(gca,'XTickLabel',v)
```

```
subplot(2,3,6)
hold on
title('Lokal kraft v\"anster kn\"a', 'Interpreter', 'LaTex')
plot(1:D_Frames, Segments(index_seg('lshank')).local_force(3,:))
xlabel('Tid (s)')
ylabel('Kraft i \zeta-led (N)')
set(gca,'XTickLabel',v)
```

B.9 plot_movements

```
function plot_movements(AllPointsMotion, Header, Frames,Segments,...
    index_segments,lab,static_or_dynamic,lines_or_None,dots_or_None,...
    LCS_or_None,force_or_None,All)
% -------------------------------------------------------------------------
% Input =   AllPointsMotion: Structure containing all markers motion
%           Header: a cell containing all markers names.
%           Frames: Number of samples.
%           Segments: A data structure containing segment parts and
%           corresponding data.
%           index_segments: index for segments.
%           lab : If the measurements were done at 'GU' or 'Qualisys' lab.
%           static_or_dynamic_as_a_string: If plot_movements shall
%           interpret the data as 'static' or 'dynamic'.
%           lines_or_None : If lines should be displayed in plot.
%           dots_or_None : If markers should be displayed in plot.
%           LCS_or_None  : If LCS should be displayed in plot.
%           force_or_None : If force should be displayed in plot.
%           All : If all avaiable data should be displayed in plot.
%
%
% Output =  None.
%
% The function plots, depending on active choice in input, a 3D-plot of
% test person as a static plot or moving dynamic plot.
% -------------------------------------------------------------------------

% Setting axis and window depending on which lab the calculations were
% collected and if the input data are static or dynamic.
if strcmp('GU',lab)
    if strcmp('static',static_or_dynamic)
        frames = 1;
        axis_scale = [800 1600 -200 1000 -50 2000];
        T = 0;
        forceplates = 0;
        view1=30;
        view2=30;
    elseif strcmp('dynamic',static_or_dynamic)
        frames = 1:floor(400/20):Frames;
        axis_scale = [200 2000 -2000 2500 -50 2000];
        % T is how long (in seconds) the plot will hold in every
        % loop before next time step plots.
        T = 0.0001;
        forceplates = 0;
        view1=90;
        view2=25;
    else
        msg = 'Error occurred - Incorrect input.';
        error(msg)
    end
elseif strcmp('Qualisys',lab)
    if strcmp('static',static_or_dynamic)
```

```
        frames = 1;
        axis_scale = [0 600 -500 100 0 1800];
        T = 0;
        view1=30;
        view2=30;
        forceplates = 2;
    elseif strcmp('dynamic',static_or_dynamic)
        frames = 1:floor(400/100):Frames;
        axis_scale = [-3000 1500 -500 1000 -50 2000];
        T = 0.0001;
        view1=30;
        view2=30;
        forceplates = 2;
    else
        msg = 'Error occurred - Incorrect input.';
        error(msg)
    end
end

for d=frames
% Collecting all coordinates
    for i=1:length(Header)
        coord(d,:,i)=AllPointsMotion(i).PosTime(d,:);
    end
% Collecting all coordinates for each bodypart
for i=1:length(AllPointsMotion);
    if strcmp(AllPointsMotion(i).Bodypart,'right foot');
        j = 1;
        while j <= length(AllPointsMotion(i).PlotRank);
                coord_right_foot(d,:,AllPointsMotion(i).PlotRank(j))= ...
                    AllPointsMotion(i).PosTime(d,:);
                j = j+1;
        end
    elseif strcmp(AllPointsMotion(i).Bodypart,'left foot');
        j = 1;
        while j <= length(AllPointsMotion(i).PlotRank);
                coord_left_foot(d,:,AllPointsMotion(i).PlotRank(j))= ...
                    AllPointsMotion(i).PosTime(d,:);
                j = j+1;
        end
     elseif strcmp(AllPointsMotion(i).Bodypart,'left shank');
        j = 1;
        while j <= length(AllPointsMotion(i).PlotRank);
                coord_left_shank(d,:,AllPointsMotion(i).PlotRank(j)) = ...
                    AllPointsMotion(i).PosTime(d,:);
                j = j+1;
        end
     elseif strcmp(AllPointsMotion(i).Bodypart,'right shank');
        j = 1;
        while j <= length(AllPointsMotion(i).PlotRank);
                coord_right_shank(d,:,AllPointsMotion(i).PlotRank(j)) = ...
                    AllPointsMotion(i).PosTime(d,:);
                j = j+1;
        end
     elseif strcmp(AllPointsMotion(i).Bodypart,'left thigh');
        j = 1;
```

```
        while j <= length(AllPointsMotion(i).PlotRank);
                coord_left_thigh(d,:,AllPointsMotion(i).PlotRank(j)) = ...
                    AllPointsMotion(i).PosTime(d,:);
                j = j+1;
        end
    elseif strcmp(AllPointsMotion(i).Bodypart,'right thigh');
        j = 1;
        while j <= length(AllPointsMotion(i).PlotRank);
                coord_right_thigh(d,:,AllPointsMotion(i).PlotRank(j)) = ...
                    AllPointsMotion(i).PosTime(d,:);
                j = j+1;
        end
    elseif strcmp(AllPointsMotion(i).Bodypart,'pelvis');
        j = 1;
        while j <= length(AllPointsMotion(i).PlotRank);
                coord_pelvis(d,:,AllPointsMotion(i).PlotRank(j)) = ...
                    AllPointsMotion(i).PosTime(d,:);
                j = j+1;
        end
    elseif strcmp(AllPointsMotion(i).Bodypart,'upper body') || ...
            strcmp(AllPointsMotion(i).Bodypart,'left arm') || ...
            strcmp(AllPointsMotion(i).Bodypart,'right arm')
        j = 1;
        while j <= length(AllPointsMotion(i).PlotRank);
                coord_whole_upper_body...
                    (d,:,AllPointsMotion(i).PlotRank(j)) = ...
                    AllPointsMotion(i).PosTime(d,:);
                j = j+1;
        end
    elseif strcmp(AllPointsMotion(i).Bodypart,'Stick')
        j = 1;
        while j <= length(AllPointsMotion(i).PlotRank);
                coord_stick(d,:,AllPointsMotion(i).PlotRank(j)) = ...
                    AllPointsMotion(i).PosTime(d,:);
                j = j+1;
        end
    elseif strcmp(AllPointsMotion(i).Bodypart,'Global')
        j = 1;
        while j <= length(AllPointsMotion(i).PlotRank);
                coord_global(d,:,AllPointsMotion(i).PlotRank(j)) = ...
                    AllPointsMotion(i).COPglobal(d,:);
                j = j+1;
        end
    end
end
end
% Collecting all coordinate lines
if exist('coord_stick','var') == 0
    coord_lines={coord_right_foot,coord_left_foot,coord_left_shank,...
    coord_right_shank,coord_left_thigh,coord_right_thigh,coord_pelvis,...
    coord_whole_upper_body};
else
    coord_lines={coord_right_foot,coord_left_foot,coord_left_shank,...
    coord_right_shank,coord_left_thigh,coord_right_thigh,coord_pelvis,...
    coord_whole_upper_body,coord_stick};
end
```

```
for d=frames
% Plotting the coordinates
    clf
    figure(1)
    axis equal
    grid on
    xlabel('x-axel')
    ylabel('y-axel')
    zlabel('z-axel')
    view(view1,view2)
    axis(axis_scale)
    hold on

    %Plotting all dots
    if strcmp('dots',dots_or_None) || strcmp('All',All)
        for i=1:length(AllPointsMotion) - forceplates;
            plot3(coord(d,1,i),coord(d,2,i), coord(d,3,i), 'ro')
        end
    end
    % If dynamic, the function also plots global coordinates
    if strcmp('dynamic',static_or_dynamic)
        for i =1:2
         plot3(coord_global(d,1,i),coord_global(d,2,i),...
             coord_global(d,3,i), 'go');
        end
    end
% Plotting all lines
    if strcmp('lines',lines_or_None)|| strcmp('All',All)
        for i=1:length(coord_lines);
            plot_coord = coord_lines{i};
            for j=1:size(coord_lines{i},3)-1;
                plot3([plot_coord(d,1,j),plot_coord(d,1,j+1)],...
                    [plot_coord(d,2,j),...
                plot_coord(d,2,j+1)],[plot_coord(d,3,j),...
                plot_coord(d,3,j+1)] ,'b', 'linewidth', 1)

            end
        end
    end

    %Plotting the segments LCS
    if strcmp('LCS',LCS_or_None) || strcmp('All',All)
        for i = 1:length(Segments)
            plot3(Segments(i).Origin(d,1),Segments(i).Origin(d,2),...
                Segments(i).Origin(d,3), 'ko')
            plot3([Segments(i).Origin(d,1), ...
                Segments(i).X(d,1)],[Segments(i).Origin(d,2), ...
                Segments(i).X(d,2)],[Segments(i).Origin(d,3),...
                Segments(i).X(d,3)],'k');
            plot3([Segments(i).Origin(d,1),...
                Segments(i).Y(d,1)],[Segments(i).Origin(d,2),...
                Segments(i).Y(d,2)],[Segments(i).Origin(d,3),...
                Segments(i).Y(d,3)],'g');
            plot3([Segments(i).Origin(d,1),...
                Segments(i).Z(d,1)],[Segments(i).Origin(d,2),...
```

```
                Segments(i).Z(d,2)],[Segments(i).Origin(d,3),...
                Segments(i).Z(d,3)],'r');
            if strcmp('dynamic',static_or_dynamic)
            plot3(Segments(i).COM(d,1),Segments(i).COM(d,2),...
                Segments(i).COM(d,3), 'ko')
            end
        end
    end

    if strcmp('force',force_or_None) || strcmp('All',All)
        plot3([Segments(index_segments('lfoot')).Point(d,1)...
            Segments(index_segments('lfoot')).Point(d,1)...
            + Segments(index_segments('lfoot')).force(d,1)],...
            [Segments(index_segments('lfoot')).Point(d,2) ...
            Segments(index_segments('lfoot')).Point(d,2)...
            + Segments(index_segments('lfoot')).force(d,2)],...
            [Segments(index_segments('lfoot')).Point(d,3)...
            Segments(index_segments('lfoot')).Point(d,3)...
            + Segments(index_segments('lfoot')).force(d,3)],...
            'r','linewidth',2)
        plot3([Segments(index_segments('rfoot')).Point(d,1)...
            Segments(index_segments('rfoot')).Point(d,1)...
            + Segments(index_segments('rfoot')).force(d,1)],...
            [Segments(index_segments('rfoot')).Point(d,2)...
            Segments(index_segments('rfoot')).Point(d,2)...
            + Segments(index_segments('rfoot')).force(d,2)]...
            ,[Segments(index_segments('rfoot')).Point(d,3)...
            Segments(index_segments('rfoot')).Point(d,3)...
            + Segments(index_segments('rfoot')).force(d,3)],...
            'r','linewidth',2)
        plot3([Segments(index_segments('lshank')).Point(d,1)...
            Segments(index_segments('lshank')).Point(d,1)...
            + Segments(index_segments('lshank')).force(d,1)],...
            [Segments(index_segments('lshank')).Point(d,2)...
            Segments(index_segments('lshank')).Point(d,2)...
            + Segments(index_segments('lshank')).force(d,2)]...
            ,[Segments(index_segments('lshank')).Point(d,3)...
            Segments(index_segments('lshank')).Point(d,3)...
            + Segments(index_segments('lshank')).force(d,3)],...
            'r','linewidth',2)
        plot3([Segments(index_segments('rshank')).Point(d,1)...
            Segments(index_segments('rshank')).Point(d,1)...
            + Segments(index_segments('rshank')).force(d,1)],...
            [Segments(index_segments('rshank')).Point(d,2)...
            Segments(index_segments('rshank')).Point(d,2)...
            + Segments(index_segments('rshank')).force(d,2)]...
            ,[Segments(index_segments('rshank')).Point(d,3)...
            Segments(index_segments('rshank')).Point(d,3)...
            + Segments(index_segments('rshank')).force(d,3)],...
            'r','linewidth',2)
    end
pause(T)
end
```

B.10 plot_dynamic_angles

```
function plot_dynamic_angles( D_Frames, dynamic_angles )
% -------------------------------------------------------------------------
% Input  = D_Frames : Number of samples
%          dynamic_angles : Angles from the dynamic movements
%
% Output = None
%
% The function plots dynamic angles for each segment
% in relation to the adjacent segment.
% -------------------------------------------------------------------------

up = ceil(D_Frames*4)/4;
down = floor(D_Frames*4)/4;
for i = 1:size(dynamic_angles,3)
    for j = 1:3
        if max(dynamic_angles(:,j,i)) > 160
            dynamic_angles(:,j,i) =  dynamic_angles(:,j,i) - 180;
        end
    end
end

if D_Frames - down < up - D_Frames;
    timestop = down/400;
else
    timestop = up/400;
end
v = [0];
j =1;
for i = 1:0.25:timestop+1;

    v(end+1) = v(j)+0.25;
    j =j+1;
end

figure(1)
set(gcf, 'name', 'Relativa vinklar')

subplot(4,2,[1 2])
hold on
title('B\"acken i f\"orh\aa{}llande till br\"ostkorg', 'Interpreter',...
    'LaTex','FontSize',14)
plot(1:D_Frames,dynamic_angles(:,1,1),'g')
plot(1:D_Frames,dynamic_angles(:,2,1),'g--')
plot(1:D_Frames,dynamic_angles(:,3,1),'g:')

legend('Rotation kring \xi-axeln','Rotation kring \eta-axeln',...
    'Rotation kring \zeta-axeln','Location','eastoutside')
xlabel('Tid (s)')
ylabel('Vinkel (grader)')
set(gca,'XTickLabel',v)
```

```
subplot(4,2,3)
hold on
title('B\"acken i f\"orh\aa{}llande till v\"anster l\aa{}r',...
    'Interpreter', 'LaTex','FontSize',14)
plot(1:D_Frames,dynamic_angles(:,1,2),'g')
plot(1:D_Frames,dynamic_angles(:,2,2),'g--')
plot(1:D_Frames,dynamic_angles(:,3,2),'g:')

xlabel('Tid (s)')
ylabel('Vinkel (grader)')
set(gca,'XTickLabel',v)

subplot(4,2,4)
hold on
title('B\"acken i f\"orh\aa{}llande till h\"oger l\aa{}r',...
    'Interpreter', 'LaTex','FontSize',14)
plot(1:D_Frames,dynamic_angles(:,1,3),'g')
plot(1:D_Frames,dynamic_angles(:,2,3),'g--')
plot(1:D_Frames,dynamic_angles(:,3,3),'g:')

xlabel('Tid (s)')
ylabel('Vinkel (grader)')
set(gca,'XTickLabel',v)


subplot(4,2,5)
hold on
title('V\"anster l\aa{}r i f\"orh\aa{}llande till v\"anster skenben',...
    'Interpreter', 'LaTex','FontSize',14)
plot(1:D_Frames,dynamic_angles(:,1,4),'g')
plot(1:D_Frames,dynamic_angles(:,2,4),'g--')
plot(1:D_Frames,dynamic_angles(:,3,4),'g:')

xlabel('Tid (s)')
ylabel('Vinkel (grader)')
ylim([-150 50])
set(gca,'XTickLabel',v)

subplot(4,2,6)
hold on
title('H\"oger l\aa{}r i f\"orh\aa{}llande till h\"oger skenben',...
    'Interpreter', 'LaTex','FontSize',14)
plot(1:D_Frames,dynamic_angles(:,1,5),'g')
plot(1:D_Frames,dynamic_angles(:,2,5),'g--')
plot(1:D_Frames,dynamic_angles(:,3,5),'g:')

xlabel('Tid (s)')
ylabel('Vinkel (grader)')
ylim([-150 50])
set(gca,'XTickLabel',v)

subplot(4,2,7)
hold on
title('V\"anster skenben i f\"orh\aa{}llande till v\"anster fot',...
    'Interpreter', 'LaTex','FontSize',14)
plot(1:D_Frames,dynamic_angles(:,1,6),'g')
```

```
plot(1:D_Frames,dynamic_angles(:,2,6),'g--')
plot(1:D_Frames,dynamic_angles(:,3,6),'g:')

xlabel('Tid (s)')
ylabel('Vinkel (grader)')
ylim([-50 50])
set(gca,'XTickLabel',v)

subplot(4,2,8)
hold on
title('H\"oger skenben i f\"orh\aa{}llande till h\"oger fot',...
    'Interpreter', 'LaTex','FontSize',14)
plot(1:D_Frames,dynamic_angles(:,1,7),'g')
plot(1:D_Frames,dynamic_angles(:,2,7),'g--')
plot(1:D_Frames,dynamic_angles(:,3,7),'g:')

xlabel('Tid (s)')
ylabel('Vinkel (grader)')
ylim([-50 50])
set(gca,'XTickLabel',v)

end
```