Chapter 7   APPENDIX

7.1 Mathematical Proofs

7.1.1 Moments of Multivariate Normal

The probability analysis presented in this thesis relies on determining the moments of the multivariate normal probability distribution. A multivariate normal deviate has the probability density function of

(258)
$$f_N\left(x;\mu,\Sigma\right) = \frac{1}{\sqrt{\left(2\pi\right)^N |\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

for a deviate x defined by

(259)
$$x = \begin{pmatrix} x_1 & x_2 & \cdots & x_N \end{pmatrix}^T$$

and the distribution parameters of

(260)
$$\mu = \begin{pmatrix} \mu_1 & \mu_2 & \cdots & \mu_N \end{pmatrix}^T$$

(261)
$$\Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} & \cdots & \Sigma_{1N} \\ \Sigma_{21} & \Sigma_{22} & \cdots & \Sigma_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{N1} & \Sigma_{N2} & \cdots & \Sigma_{NN} \end{pmatrix}$$

The moment generating function is

(262)
$$M_N\left(t;\mu,\Sigma\right) = e^{\mu^T t + \frac{1}{2}t^T \Sigma t}$$

where

(263)
$$t = \begin{pmatrix} t_1 & t_2 & \cdots & t_N \end{pmatrix}^T$$

The moments can be retrieved from the derivatives of the moment generating function by

$$(264) \qquad E\left(x_i^k x_j^l x_k^m \cdots\right) = \frac{\partial^k}{\partial t_i^k} \frac{\partial^l}{\partial t_j^l} \frac{\partial^m}{\partial t_k^m} \cdots M_N \bigg|_{t=0}$$

That leads to the moments of the elements of the vector x

$$(265) \qquad E\left(x_i\right) = \mu_i$$

$$(266) \qquad E\left(x_i x_j\right) = \mu_i \mu_j + \Sigma_{ij}$$

$$(267) \qquad E\left(x_i x_j x_k\right) = \mu_i \mu_j \mu_k + \Sigma_{ij}\mu_k + \Sigma_{ik}\mu_j + \Sigma_{jk}\mu_i$$

$$(268) \qquad \begin{aligned} E\left(x_i x_j x_k x_l\right) &= \mu_i \mu_j \mu_k \mu_l + \Sigma_{ij}\Sigma_{kl} + \Sigma_{ik}\Sigma_{lj} + \Sigma_{il}\Sigma_{jk} \\ &+ \Sigma_{ij}\mu_k\mu_l + \Sigma_{ik}\mu_j\mu_l + \Sigma_{il}\mu_j\mu_k + \Sigma_{jk}\mu_i\mu_l + \Sigma_{jl}\mu_i\mu_k + \Sigma_{kl}\mu_i\mu_j \end{aligned}$$

$$(269) \qquad \begin{aligned} E\left(x_i x_j x_k x_l x_m\right) &= \mu_i \mu_j \mu_k \mu_l \mu_m \\ &+ \Sigma_{ij}\left(\mu_k\mu_l\mu_m + \Sigma_{kl}\mu_m + \Sigma_{km}\mu_l + \Sigma_{lm}\mu_k\right) \\ &+ \Sigma_{ik}\left(\mu_j\mu_l\mu_m + \Sigma_{jl}\mu_m + \Sigma_{jm}\mu_l + \Sigma_{lm}\mu_j\right) \\ &+ \Sigma_{il}\left(\mu_j\mu_k\mu_m + \Sigma_{jk}\mu_m + \Sigma_{jm}\mu_k + \Sigma_{km}\mu_j\right) \\ &+ \Sigma_{im}\left(\mu_j\mu_k\mu_l + \Sigma_{jk}\mu_l + \Sigma_{jl}\mu_k + \Sigma_{kl}\mu_j\right) \\ &+ \Sigma_{jk}\mu_i\mu_l\mu_m + \Sigma_{jl}\mu_i\mu_k\mu_m + \Sigma_{jm}\mu_i\mu_k\mu_l \\ &+ \Sigma_{jm}\Sigma_{kl}\mu_i + \Sigma_{jk}\Sigma_{lm}\mu_i + \Sigma_{jl}\Sigma_{mk}\mu_i \\ &+ \Sigma_{kl}\mu_i\mu_j\mu_m + \Sigma_{km}\mu_i\mu_j\mu_l + \Sigma_{lm}\mu_i\mu_j\mu_k \end{aligned}$$

$$E\left(x_i x_j x_k x_l x_m x_n\right) = \mu_i \mu_j \mu_k \mu_l \mu_m \mu_n$$

$$+\Sigma_{ij}\begin{pmatrix} \mu_m \mu_n \mu_k \mu_l + \Sigma_{mn}\Sigma_{kl} + \Sigma_{mk}\Sigma_{nl} + \Sigma_{kn}\Sigma_{lm} \\ +\Sigma_{kl}\mu_m \mu_n + \Sigma_{km}\mu_l \mu_n + \Sigma_{kn}\mu_l \mu_m + \Sigma_{lm}\mu_k \mu_n \\ +\Sigma_{nl}\mu_k \mu_m + \Sigma_{mn}\mu_k \mu_l \end{pmatrix}$$

$$+\Sigma_{ik}\begin{pmatrix} \mu_j \mu_l \mu_m \mu_n + \Sigma_{jl}\Sigma_{mn} + \Sigma_{jm}\Sigma_{nl} + \Sigma_{jn}\Sigma_{lm} \\ +\Sigma_{jl}\mu_m \mu_n + \Sigma_{jm}\mu_l \mu_n + \Sigma_{jn}\mu_l \mu_m + \Sigma_{lm}\mu_j \mu_n \\ +\Sigma_{nl}\mu_j \mu_m + \Sigma_{mn}\mu_j \mu_l \end{pmatrix}$$

(270)

$$+\Sigma_{il}\begin{pmatrix} \mu_j \mu_k \mu_m \mu_n + \Sigma_{jk}\Sigma_{mn} + \Sigma_{jm}\Sigma_{kn} + \Sigma_{jn}\Sigma_{km} \\ +\Sigma_{jk}\mu_m \mu_n + \Sigma_{jm}\mu_k \mu_n + \Sigma_{jn}\mu_k \mu_m + \Sigma_{km}\mu_j \mu_n \\ +\Sigma_{kn}\mu_j \mu_m + \Sigma_{mn}\mu_j \mu_k \end{pmatrix}$$

$$+\Sigma_{im}\begin{pmatrix} \mu_j \mu_k \mu_l \mu_n + \Sigma_{jk}\Sigma_{nl} + \Sigma_{jl}\Sigma_{kn} + \Sigma_{jn}\Sigma_{kl} \\ +\Sigma_{jk}\mu_l \mu_n + \Sigma_{jl}\mu_k \mu_n + \Sigma_{jn}\mu_k \mu_l + \Sigma_{kl}\mu_j \mu_n \\ +\Sigma_{kn}\mu_j \mu_l + \Sigma_{nl}\mu_j \mu_k \end{pmatrix}$$

$$+\Sigma_{in}\begin{pmatrix} \mu_j \mu_k \mu_l \mu_m + \Sigma_{jk}\Sigma_{lm} + \Sigma_{jl}\Sigma_{km} + \Sigma_{jm}\Sigma_{kl} \\ +\Sigma_{jk}\mu_l \mu_m + \Sigma_{jl}\mu_k \mu_m + \Sigma_{jm}\mu_k \mu_l + \Sigma_{kl}\mu_j \mu_m \\ +\Sigma_{km}\mu_j \mu_l + \Sigma_{lm}\mu_j \mu_k \end{pmatrix}$$

$$+\mu_i\begin{pmatrix} \Sigma_{jk}\left(\mu_l \mu_m \mu_n + \Sigma_{lm}\mu_n + \Sigma_{ln}\mu_m + \Sigma_{mn}\mu_l\right) \\ +\Sigma_{jl}\left(\mu_k \mu_m \mu_n + \Sigma_{km}\mu_n + \Sigma_{kn}\mu_m + \Sigma_{mn}\mu_k\right) \\ +\Sigma_{jm}\left(\mu_k \mu_l \mu_n + \Sigma_{kl}\mu_n + \Sigma_{kn}\mu_l + \Sigma_{ln}\mu_k\right) \\ +\Sigma_{jn}\left(\mu_k \mu_l \mu_m + \Sigma_{kl}\mu_m + \Sigma_{km}\mu_l + \Sigma_{lm}\mu_k\right) \\ +\Sigma_{kl}\mu_j \mu_m \mu_n + \Sigma_{km}\mu_j \mu_l \mu_n + \Sigma_{kn}\mu_j \mu_l \mu_m \\ +\Sigma_{kn}\Sigma_{lm}\mu_j + \Sigma_{kl}\Sigma_{mn}\mu_j + \Sigma_{km}\Sigma_{nl}\mu_j \\ +\Sigma_{lm}\mu_j \mu_k \mu_n + \Sigma_{ln}\mu_j \mu_k \mu_m + \Sigma_{mn}\mu_j \mu_k \mu_l \end{pmatrix}$$

If all indices are equal, the expectations simplify to

(271)
$$E\left(x_i\right) = \mu_i$$

(272)
$$E\left(x_i^2\right) = \mu_i^2 + \Sigma_{ii}$$

(273)
$$E\left(x_i^3\right) = \mu_i^3 + 3\Sigma_{ii}\mu_i$$

(274)
$$E\left(x_i^4\right) = \mu_i^4 + 6\Sigma_{ii}\mu_i^2 + 3\Sigma_{ii}^2$$

(275)
$$E\left(x_i^5\right) = \mu_i^5 + 10\Sigma_{ii}\mu_i^3 + 15\Sigma_{ii}^2\mu_i$$

(276)
$$E\left(x_i^6\right) = \mu_i^6 + 15\Sigma_{ii}\mu_i^4 + 45\Sigma_{ii}^2\mu_i^2 + 15\Sigma_{ii}^3$$

The vector moments are

(277)
$$E(x) = \mu$$

(278)
$$E\left(xx^T\right) = \mu\mu^T + \Sigma$$

(279)
$$E\left(x^T x\right) = \mu^T\mu + Tr(\Sigma)$$

(280)
$$E\left(xx^T x\right) = \mu\mu^T\mu + Tr(\Sigma)\mu + 2\Sigma\mu$$

(281)
$$E\left(xx^T xx^T\right) = \mu\mu^T\mu\mu^T + 2\Sigma^2 + \Sigma Tr(\Sigma) + Tr(\Sigma)\mu\mu^T + 2\mu\mu^T\Sigma + 2\Sigma\mu\mu^T + \Sigma\mu^T\mu$$

(282)
$$E\left(xx^T ax^T\right) = \mu\mu^T a\mu^T + \Sigma\mu^T a + \Sigma a\mu^T + \mu a^T\Sigma$$

(283)
$$E\left(x^T xx^T x\right) = \mu^T\mu\mu^T\mu + 2Tr\left(\Sigma^2\right) + Tr(\Sigma)^2 + 2Tr(\Sigma)\mu^T\mu + 4\mu^T\Sigma\mu$$

(284)
$$E\left(xx^T xx^T x\right) = \mu\mu^T\mu\mu^T\mu + 2Tr(\Sigma)\mu\mu^T\mu + 4\mu\mu^T\Sigma\mu + 4\Sigma\mu\mu^T\mu$$
$$+2Tr\left(\Sigma^2\right)\mu + Tr(\Sigma)^2\mu + 4Tr(\Sigma)\Sigma\mu + 8\Sigma^2\mu$$

(285)
$$E\left(xx^T xx^T ax^T\right) = \mu\mu^T\mu\mu^T a\mu^T + \Sigma a^T\mu\mu^T\mu + \Sigma Tr(\Sigma)a^T\mu + Tr(\Sigma)\mu\mu^T a\mu^T$$
$$+2\Sigma a^T\Sigma\mu + \Sigma a\mu^T\mu\mu^T + 2\Sigma a\mu^T\Sigma + \Sigma a\mu^T Tr(\Sigma) + 2\Sigma\mu a^T\mu\mu^T$$
$$+2\Sigma\mu a^T\Sigma + 2\Sigma\Sigma a^T\mu + 2\Sigma\Sigma a\mu^T + \mu\mu^T\mu a^T\Sigma + 2\mu a^T\mu\mu^T\Sigma$$
$$+2\mu a^T\Sigma\Sigma + \mu a^T\Sigma Tr(\Sigma) + 2\mu a^T\Sigma\mu\mu^T$$

(286)
$$E\left(xx^T ab^T xx^T\right) = \mu\mu^T ab^T\mu\mu^T + \Sigma a^T\Sigma b + \Sigma ab^T\Sigma + \Sigma ba^T\Sigma + \Sigma\mu^T ab^T\mu$$
$$+\Sigma ab^T\mu\mu^T + \Sigma ba^T\mu\mu^T + \mu\mu^T ba^T\Sigma + \mu\mu^T ab^T\Sigma + \mu b^T\Sigma a\mu^T$$

(287)
$$E\left(xx^T Axx^T\right) = \mu\mu^T A\mu\mu^T + \Sigma Tr(A\Sigma) + \Sigma A\Sigma + \Sigma A^T\Sigma + \Sigma\mu^T A\mu$$
$$+\Sigma A\mu\mu^T + \Sigma A^T\mu\mu^T + \mu\mu^T A^T\Sigma + \mu\mu^T A\Sigma + Tr\left(A^T\Sigma\right)\mu\mu^T$$

$$E\left(xx^T xx^T xx^T\right) = \mu\mu^T \mu\mu^T \mu\mu^T + \Sigma Tr(\Sigma)^2 + 2\Sigma Tr\left(\Sigma^2\right) + 8\Sigma^3 + 4\Sigma^2 Tr(\Sigma)$$
$$+ 4\Sigma\mu^T \Sigma\mu + 8\Sigma\mu\mu^T \Sigma + 4\Sigma^2 \mu^T \mu + 8\Sigma^2 \mu\mu^T + 8\mu\mu^T \Sigma^2$$
$$+ 4\mu\mu^T \Sigma Tr(\Sigma) + 4\Sigma Tr(\Sigma)\mu\mu^T + 2Tr\left(\Sigma^2\right)\mu\mu^T + Tr(\Sigma)^2 \mu\mu^T$$
$$+ 2\Sigma Tr(\Sigma)\mu^T \mu + 4\Sigma\mu\mu^T \mu\mu^T + 4\mu\mu^T \mu\mu^T \Sigma + 4\mu\mu^T \Sigma\mu\mu^T$$
$$+ 2\mu\mu^T \mu\mu^T Tr(\Sigma) + \Sigma\mu^T \mu\mu^T \mu$$

(288)

The vector covariances are calculated with the definition of

(289)
$$Cov(a,b) = E(ab) - E(a)E(b)$$

to produce the following

(290)
$$Cov\left(x,x^T\right) = \Sigma$$

(291)
$$Cov\left(x^T,x\right) = Tr(\Sigma)$$

(292)
$$Cov\left(x,x^T x\right) = 2\Sigma\mu$$

(293)
$$Cov\left(xx^T,x\right) = Tr(\Sigma)\mu + \Sigma\mu$$

(294)
$$Cov\left(x,x^T xx^T\right) = 2\Sigma^2 + \Sigma Tr(\Sigma) + 2\Sigma\mu\mu^T + \Sigma\mu^T \mu$$

(295)
$$Cov\left(xx^T,xx^T\right) = \Sigma^2 + \Sigma Tr(\Sigma) + Tr(\Sigma)\mu\mu^T + \mu\mu^T \Sigma + \Sigma\mu\mu^T + \Sigma\mu^T \mu$$

(296)
$$Cov\left(xx^T x,x^T\right) = 2\Sigma^2 + \Sigma Tr(\Sigma) + 2\mu\mu^T \Sigma + \Sigma\mu^T \mu$$

(297)
$$Cov\left(x,x^T ax^T\right) = \Sigma\mu^T a + \Sigma a\mu^T$$

(298)
$$Cov\left(xx^T,ax^T\right) = \Sigma\mu^T a + \mu a^T \Sigma$$

(299)
$$Cov\left(xx^T a,x^T\right) = \Sigma\mu^T a + \mu a^T \Sigma$$

(300)
$$Cov\left(x^T,xx^T x\right) = 2Tr\left(\Sigma^2\right) + Tr(\Sigma)^2 + Tr(\Sigma)\mu^T \mu + 2\mu^T \Sigma\mu$$

(301)
$$Cov\left(x^T x,x^T x\right) = 2Tr\left(\Sigma^2\right) + 4\mu^T \Sigma\mu$$

(302)
$$Cov\left(x^T xx^T,x\right) = 2Tr\left(\Sigma^2\right) + Tr(\Sigma)^2 + Tr(\Sigma)\mu^T \mu + 2\mu^T \Sigma\mu$$

$$(303) \qquad Cov\left(x, x^T x x^T x\right) = 4\Sigma\mu\mu^T\mu + 4Tr(\Sigma)\Sigma\mu + 8\Sigma^2\mu$$

$$(304) \qquad Cov\left(xx^T, xx^T x\right) = \left(\left(Tr(\Sigma) + 3\Sigma\right)\left(\mu\mu^T + Tr(\Sigma)\right) + 2\mu\mu^T\Sigma + 2Tr\left(\Sigma^2\right) + 6\Sigma^2\right)\mu$$

$$(305) \qquad Cov\left(xx^T x, x^T x\right) = 4\mu\mu^T\Sigma\mu + 2\Sigma\mu\mu^T\mu + 2Tr\left(\Sigma^2\right)\mu + 2Tr(\Sigma)\Sigma\mu + 8\Sigma^2\mu$$

$$(306)$$
$$Cov\left(xx^T xx^T, x\right) = Tr(\Sigma)\mu\mu^T\mu + 2\mu\mu^T\Sigma\mu + \Sigma\mu\mu^T\mu$$
$$+2Tr\left(\Sigma^2\right)\mu + Tr(\Sigma)^2\mu + 3Tr(\Sigma)\Sigma\mu + 6\Sigma^2\mu$$

$$(307)$$
$$Cov\left(x, x^T xx^T xx^T\right) = \Sigma Tr(\Sigma)^2 + 2\Sigma Tr\left(\Sigma^2\right) + 8\Sigma^3 + 4\Sigma^2 Tr(\Sigma)$$
$$+4\Sigma\mu^T\Sigma\mu + 8\Sigma\mu\mu^T\Sigma + 4\Sigma^2\mu^T\mu + 8\Sigma^2\mu\mu^T$$
$$+4\Sigma Tr(\Sigma)\mu\mu^T + 2\Sigma Tr(\Sigma)\mu^T\mu + 4\Sigma\mu\mu^T\mu\mu^T + \Sigma\mu^T\mu\mu^T\mu$$

$$(308)$$
$$Cov\left(xx^T, xx^T xx^T\right) = \Sigma Tr(\Sigma)^2 + 2\Sigma Tr\left(\Sigma^2\right) + 6\Sigma^3 + 3\Sigma^2 Tr(\Sigma)$$
$$+4\Sigma\mu^T\Sigma\mu + 6\Sigma\mu\mu^T\Sigma + 3\Sigma^2\mu^T\mu + 6\Sigma^2\mu\mu^T + 6\mu\mu^T\Sigma^2$$
$$+4\mu\mu^T\Sigma Tr(\Sigma) + 3\Sigma Tr(\Sigma)\mu\mu^T + 2Tr\left(\Sigma^2\right)\mu\mu^T + Tr(\Sigma)^2\mu\mu^T$$
$$+2\Sigma Tr(\Sigma)\mu^T\mu + 3\Sigma\mu\mu^T\mu\mu^T + \mu\mu^T\mu\mu^T\Sigma + 2\mu\mu^T\Sigma\mu\mu^T$$
$$+\mu\mu^T\mu\mu^T Tr(\Sigma) + \Sigma\mu^T\mu\mu^T\mu$$

$$(309)$$
$$Cov\left(xx^T x, x^T xx^T\right) = \Sigma Tr(\Sigma)^2 + 2\Sigma Tr\left(\Sigma^2\right) + 8\Sigma^3 + 4\Sigma^2 Tr(\Sigma)$$
$$+4\Sigma\mu^T\Sigma\mu + 4\Sigma\mu\mu^T\Sigma + 4\Sigma^2\mu^T\mu + 8\Sigma^2\mu\mu^T + 8\mu\mu^T\Sigma^2$$
$$+2\mu\mu^T\Sigma Tr(\Sigma) + 2\Sigma Tr(\Sigma)\mu\mu^T + 2Tr\left(\Sigma^2\right)\mu\mu^T + 2\Sigma Tr(\Sigma)\mu^T\mu$$
$$+2\Sigma\mu\mu^T\mu\mu^T + 2\mu\mu^T\mu\mu^T\Sigma + 4\mu\mu^T\Sigma\mu\mu^T + \Sigma\mu^T\mu\mu^T\mu$$

The vector central moments are

$$(310) \qquad E(x - \mu) = 0$$

$$(311) \qquad E\left((x - \mu)(x - \mu)^T\right) = \Sigma$$

$$(312) \qquad Cov\left((x - \mu),(x - \mu)^T\right) = \Sigma$$

$$(313) \qquad E\left((x - \mu)^T(x - \mu)\right) = Tr(\Sigma)$$

$$(314) \qquad Cov\left((x - \mu)^T,(x - \mu)\right) = Tr(\Sigma)$$

$$(315) \qquad E\Big((x-\mu)(x-\mu)^T(x-\mu)\Big)=0$$

$$(316) \qquad E\Big((x-\mu)(x-\mu)^T(x-\mu)(x-\mu)^T\Big)=2\Sigma^2+\Sigma Tr(\Sigma)$$

$$(317) \qquad Cov\Big((x-\mu)(x-\mu)^T,(x-\mu)(x-\mu)^T\Big)=\Sigma^2+\Sigma Tr(\Sigma)$$

$$(318) \qquad E\Big((x-\mu)^T(x-\mu)(x-\mu)^T(x-\mu)\Big)=2Tr(\Sigma^2)+Tr(\Sigma)^2$$

$$(319) \qquad Cov\Big((x-\mu)^T(x-\mu),(x-\mu)^T(x-\mu)\Big)=2Tr(\Sigma^2)$$

$$(320) \qquad E\Big((x-\mu)(x-\mu)^T(x-\mu)(x-\mu)^T(x-\mu)\Big)=0$$

$$(321) \qquad \begin{aligned} E\Big((x-\mu)(x-\mu)^T(x-\mu)(x-\mu)^T(x-\mu)(x-\mu)^T\Big)&=8\Sigma^3+4\Sigma^2 Tr(\Sigma)\\ &+2\Sigma Tr(\Sigma^2)+\Sigma Tr(\Sigma)^2 \end{aligned}$$

$$(322) \qquad \begin{aligned} Cov\Big((x-\mu)(x-\mu)^T(x-\mu),(x-\mu)^T(x-\mu)(x-\mu)^T\Big)&=8\Sigma^3+4\Sigma^2 Tr(\Sigma)\\ &+2\Sigma Tr(\Sigma^2)+\Sigma Tr(\Sigma)^2 \end{aligned}$$

$$(323) \quad E\Big((x-\mu)^T(x-\mu)(x-\mu)^T(x-\mu)(x-\mu)^T(x-\mu)\Big)=8Tr(\Sigma^3)+6Tr(\Sigma^2)Tr(\Sigma)+Tr(\Sigma)^3$$

Now, we will consider when there are multiple measurements with the same covariance but different mean. The kth measurement then has the properties of

$$(324) \qquad E(x_k)=\mu_k \text{ and}$$

$$(325) \qquad Cov(x_k,x_k^T)=\Sigma$$

The vector expectations of the averages of N independent measurements are

$$(326) \qquad E(\bar{x})=\bar{\mu}$$

$$(327) \qquad E(x_k\bar{x}^T)=\mu_k\bar{\mu}^T+\tfrac{1}{N}\Sigma$$

$$(328) \qquad E(x_k x_l^T\bar{x})=\mu_k\mu_l^T\bar{\mu}+\tfrac{1}{N}Tr(\Sigma)\mu_k+\tfrac{1}{N}\Sigma\mu_l$$

$$(329) \quad E\left(x_k x_k^T \bar{x}\right) = \left(\mu_k \mu_k^T + \Sigma\right)\bar{\mu} + \tfrac{1}{N} Tr(\Sigma)\mu_k + \tfrac{1}{N}\Sigma\mu_k$$

$$(330) \quad E\left(\bar{x}\bar{x}^T \bar{x}\right) = \left(\bar{\mu}\bar{\mu}^T + \tfrac{1}{N}\Sigma\right)\bar{\mu} + \tfrac{1}{N} Tr(\Sigma)\bar{\mu} + \tfrac{1}{N}\Sigma\bar{\mu}$$

$$(331) \quad E\left(x_k \bar{x}^T \bar{x}\right) = \mu_k\left(\bar{\mu}^T \bar{\mu} + \tfrac{1}{N} Tr(\Sigma)\right) + \tfrac{2}{N}\Sigma\bar{\mu}$$

$$(332) \quad E\left(\bar{x}\bar{x}^T x_k\right) = \left(\bar{\mu}\bar{\mu}^T + \tfrac{1}{N}\Sigma\right)\mu_k + \tfrac{1}{N} Tr(\Sigma)\bar{\mu} + \tfrac{1}{N}\Sigma\bar{\mu}$$

$$(333) \quad E\left(\bar{x}x_k^T x_k\right) = \bar{\mu}\left(\mu_k^T \mu_k + Tr(\Sigma)\right) + \tfrac{2}{N}\Sigma\mu_k$$

$$(334) \quad E\left(\bar{x}x_k^T x_l\right) = \bar{\mu}\mu_k^T \mu_l + \tfrac{1}{N}\Sigma\mu_k + \tfrac{1}{N}\Sigma\mu_l$$

The sample vector central moments are

$$(335) \quad E\left(x_k - \bar{x}\right) = \mu_k - \bar{\mu}$$

$$(336) \quad E\left(\left(x_k - \bar{x}\right)\left(x_l - \bar{x}\right)^T\right) = \left(\mu_k - \bar{\mu}\right)\left(\mu_l - \bar{\mu}\right)^T - \tfrac{1}{N}\Sigma$$

$$(337) \quad E\left(\left(x_k - \bar{x}\right)^T \left(x_l - \bar{x}\right)\right) = \left(\mu_k - \bar{\mu}\right)^T \left(\mu_l - \bar{\mu}\right) - \tfrac{1}{N} Tr(\Sigma)$$

$$(338) \quad \begin{aligned} E\left(\left(x_k - \bar{x}\right)\left(x_l - \bar{x}\right)^T \left(x_l - \bar{x}\right)\right) &= \left(\mu_k - \bar{\mu}\right)\left(\mu_l - \bar{\mu}\right)^T \left(\mu_l - \bar{\mu}\right) \\ &+ \tfrac{N-1}{N} Tr(\Sigma)\left(\mu_k - \bar{\mu}\right) - \tfrac{2}{N}\Sigma\left(\mu_l - \bar{\mu}\right) \end{aligned}$$

Unbiased estimators of the vector central moments from N samples are

$$(339) \quad \mathbf{C} = \tfrac{1}{N-1}\sum_{k=1}^{N}\left(x_k - \bar{x}\right)\left(x_k - \bar{x}\right)^T$$

$$(340) \quad E(\mathbf{C}) = \mathbf{C}_0 + \Sigma$$

$$(341) \quad Cov\left(\mathbf{C}_{ij}, \mathbf{C}_{mn}\right) = \tfrac{1}{N-1}\left(\Sigma_{im}\Sigma_{nj} + \Sigma_{in}\Sigma_{jm} + \Sigma_{im}\mathbf{C}_{0nj} + \Sigma_{in}\mathbf{C}_{0jm} + \Sigma_{jm}\mathbf{C}_{0in} + \Sigma_{jn}\mathbf{C}_{0im}\right)$$

$$(342) \quad D = \tfrac{1}{N-1}\sum_{k=1}^{N}\left(x_k - \bar{x}\right)^T \left(x_k - \bar{x}\right)$$

$$(343) \quad E(D) = Tr\left(\mathbf{C}_0 + \Sigma\right)$$

$$(344) \quad Cov(D,D) = \tfrac{1}{N-1} Tr\left(2\Sigma^2 + 4\mathbf{C}_0\Sigma\right)$$

$$(345) \quad \mathbf{S} = \tfrac{1}{N-1} \sum_{k=1}^{N} \left(x_k - \bar{x} \right)\left(x_k - \bar{x} \right)^T \left(x_k - \bar{x} \right)$$

$$(346) \quad E(\mathbf{S}) = \mathbf{S}_0$$

$$(347) \quad Cov\left(\mathbf{S}, \mathbf{S}^T\right) = \tfrac{1}{N-1}\left(\Sigma^3 + \Sigma^2 \mathbf{C}_0 + \Sigma \mathbf{C}_0 \Sigma + \mathbf{C}_0 \Sigma^2 + \mathbf{F}_0 \Sigma + \Sigma \mathbf{F}_0 \right)$$

$$(348) \quad \mathbf{F} = \tfrac{N}{(N-1)^2} \sum_{k=1}^{N} \left(x_k - \bar{x} \right)\left(x_k - \bar{x} \right)^T \left(x_k - \bar{x} \right)\left(x_k - \bar{x} \right)^T$$

$$(349) \quad E(\mathbf{F}) = \mathbf{F}_0 + 2\Sigma^2 + \Sigma Tr(\Sigma)$$

$$(350) \quad G = \tfrac{N}{(N-1)^2} \sum_{k=1}^{N} \left(x_k - \bar{x} \right)^T \left(x_k - \bar{x} \right)\left(x_k - \bar{x} \right)^T \left(x_k - \bar{x} \right)$$

$$(351) \quad E(G) = Tr(\mathbf{F}_0) + 2Tr\left(\Sigma^2\right) + Tr(\Sigma)^2$$

7.1.2 Positive-Semidefinite Sample Covariance

A positive-semidefinite matrix $\mathbf{A}$ is defined as having the property $h^T\mathbf{A}h \geq 0$ for an arbitrary vector $h \neq \mathbf{0}$. The sample covariance matrix is

$$(352) \quad \mathbf{C} = \tfrac{1}{N-1} \sum_{i=1}^{N} \left(x_i - \bar{x} \right)\left(x_i - \bar{x} \right)^T$$

Then applying an arbitrary vector h

$$(353) \quad h^T\mathbf{C}h = \tfrac{1}{N-1} \sum_{i=1}^{N} h^T\left(x_i - \bar{x} \right)\left(x_i - \bar{x} \right)^T h$$

Now defining $q_i \equiv \left(x_i - \bar{x} \right)^T h$ simplifies the sum to

$$(354) \quad h^T\mathbf{C}h = \tfrac{1}{N-1} \sum_{i=1}^{N} q_i^2$$

which is always positive for an arbitrary set of values q_i except for one non-trivial case. The exception is when the points are coplanar. In the coplanar case, $q_i = 0$ for all i when h is normal to the plane and therefore $h^T \mathbf{C} h = 0$.

7.2 Inertial Properties of a Tetrahedron

This paper does not deal with inertia but in the process of doing this research, various physical models were created. One model was the tetrahedral mesh along with its physical properties. Inertia properties were not easy to calculate so this section was left in for the convenience of future simulation work. Inertial properties can be quite difficult to calculate but can be exactly determined. The moment of inertia and angular momentum is taken from standard analytical mechanics books as

$$(355) \qquad\qquad I = \int \left| \hat{w} \times r \right|^2 dm$$

$$(356) \qquad\qquad L = \int r \times (w \times r) dm$$

where $dm = \rho\, dV$, the density times the differential volume.

The moment of inertia and angular momentum are very desirable traits to follow in any dynamic simulation. They both involve integrating over the tetrahedral volume. I have derived them below using parameterized coordinates. A point inside the tetrahedron can be uniquely determined by

$$(357) \qquad\qquad r = a_0 + \left(a_1 + \left(a_2 + a_3 t_0 \right) t_1 \right) t_2$$

where

$$(358) \qquad\qquad a_0 = v_2$$

$$(359) \qquad\qquad a_1 = v_3 - v_2$$

(360)
$$a_2 = v_0 - v_3$$

(361)
$$a_3 = v_1 - v_0$$

and v_i are one of the tetrahedron vertices

The differential volume is determined by the parameter space change formula

(362)
$$dV = \frac{\partial r}{\partial t_2} \cdot \left(\frac{\partial r}{\partial t_1} \times \frac{\partial r}{\partial t_0} \right) dt_0 dt_1 dt_2$$

(363)
$$\frac{\partial r}{\partial t_2} \cdot \left(\frac{\partial r}{\partial t_1} \times \frac{\partial r}{\partial t_0} \right) 6 t_1 t_2^2 V_{Tet}$$

Each parameter t_i varies from zero to one for inside the tetrahedron so the entire integral

results in

(364)
$$I = 6m \int_0^1 \int_0^1 \int_0^1 |\hat{w} \times r|^2 t_1 t_2^2 dt_0 dt_1 dt_2$$

where $\hat{w}$ is the angular velocity unit vector (i.e. the spin axis) and m is the mass of the

entire tetrahedron. The density of the tetrahedron is kept constant and thus the mass is

brought out of the integrand as the density times the volume. Performing the triple inte-

gral produces a double sum. The integral turns into the double sum

(365)
$$I = 6m \sum_{i=0}^3 \sum_{j=0}^3 (\hat{w} \times a_i) \cdot (\hat{w} \times a_j) \int_0^1 \int_0^1 \int_0^1 t_0^{n_{0ij}} t_1^{n_{1ij}} t_2^{n_{2ij}} dt_0 dt_1 dt_2$$

where

(366)
$$n_{kij} = \left\lfloor \frac{i+k}{3} \right\rfloor + \left\lfloor \frac{j+k}{3} \right\rfloor + k$$

Integrating the double sum and expanding the a_i terms into v_i terms produces the equation

$$(367) \qquad I = m\sum_{i=0}^{3}\sum_{j=0}^{3}q_{ij}a_{ij}$$

$$(368) \qquad a_{ij} = (\hat{w}\times v_i)\cdot(\hat{w}\times v_j)$$

$$(369) \qquad q_{ij} = \begin{cases} \dfrac{1}{20} & j\neq i \\ \dfrac{1}{10} & j=i \end{cases}$$

Since both q_{ij} and a_{ij} are symmetric then Equation (367) reduces to ten terms.

The angular momentum is very similar with the integral

$$(370) \qquad L = 6m\int_0^1\int_0^1\int_0^1 r\times(w\times r)t_1 t_2^2 dt_0 dt_1 dt_2$$

$$(371) \qquad L = m\sum_{i=0}^{3}\sum_{j=0}^{3}q_{ij}b_{ij}$$

$$(372) \qquad b_{ij} = v_i\times(w\times v_j)$$

In this case, b_{ij} is not symmetric so all sixteen terms must be calculated. A much simpler form of these equations occurs when a body has a coordinate system that is centered on its center of mass and is aligned to its principal axes. This simplification will be approached in the next section. The following matrix equations separate out the purely geometric quantity Q from the spinning quantity $\hat{w}$.

$$(373) \qquad I_{tet} = m_{tet}\hat{w}^T Q(c)\hat{w}$$

$$(374) \qquad L_{tet} = m_{tet}Q(c)w$$

$$(375) \qquad Q_{tet}(c) = Q(c_m - c) + \frac{1}{20}\sum_{i=0}^{3}Q(v_i - c_m)$$

$$(376) \qquad Q(\rho) = \rho^T \rho - \rho\rho^T = \begin{pmatrix} \rho^2 - x^2 & -xy & -xz \\ -yx & \rho^2 - y^2 & -yz \\ -zx & -zy & \rho^2 - z^2 \end{pmatrix}$$

The time complexity of Q_{tet} is 30 multiplications + 54 additions. For each of the above equations, the vectors are relative to the center of rotation. To generalize, one must subtract the center of rotation from the four vertex vectors.

7.3 File Formats

7.3.1 Marker Association Format

This file is a simple text file that associates the marker identification string stored in a C3D file with a known marker label. The known marker label is identified in a marker-set file that clearly associates it with a particular segment on the body. This file allows for quick association of the data with a segment. The only other alternative is to do grouping analysis for all of the data that is very time consuming. The file contains the following format:

```
46 WANDS

MARKERSET vicon512.txt

TestSubjectProdMS-V2:Root = unknown
TestSubjectProdMS-V2:LFWT = LFWT
TestSubjectProdMS-V2:RFWT = RFWT
TestSubjectProdMS-V2:LBWT = LBWT
TestSubjectProdMS-V2:RBWT = RBWT
TestSubjectProdMS-V2:LKNE = LKNE
TestSubjectProdMS-V2:LTHI = LTHI
TestSubjectProdMS-V2:LANK = LANK
TestSubjectProdMS-V2:LSHN = LLEG
TestSubjectProdMS-V2:LHEE = LHEE
…
```

7.3.2 Articulated Tetrahedral Model Format

The Articulated Tetrahedral Model (ATM) file format was created in order to bring together the necessary information to produce a jointed figure. It is a text format that references other the individual segments' mesh files. The text file has keywords followed by values and sub-fields. All words are separated by spaces. The following are the list of available keywords:

```
FIGURE myFigure
```

This keyword defines the name of the entire figure.

```
SEGMENTS 16
```

The number of segments of the figure follows:

```
MESH 0 chest.mesh
```

mesh identifies the mesh file that follows the index value. The file is identified as a tetrahedral or triangular mesh in the *.mesh format defined by the freeware MEDIT tool for editing meshes and explained in Chapter 7.3.3. The filename is for a file that is located in the same folder as the ATM file.

```
NAME Chest
```

The name of the segment follows this keyword.

```
MASS 30.0 // kg
```

The mass of the segment follows this keyword.

```
SCALE_XYZ 0.2 0.2 1.0
```

```
TRANSLATE_X 1.5
TRANSLATE_Y 1.5
TRANSLATE_Z 1.5
TRANSLATE_XYZ 1.5 2.5 3.5
ROTATE_X 35.2
PARENT 4 // hips
```

The index of the parent segment is defined here.

```
JOINT Waist
  3 0.994765 1.002953 1.084350
DEGREES_OF_FREEDOM 1 0
  VALUE 0.0
  3 -0.257042 0.966392 -0.003915
  FROM -93.199994 TO 29.000000
```

The name of the joint that is proximal to this segment is defined here followed by joint parameters. The center of rotation is defined immediately following the JOINT keyword. First a count of how many dimensions for the vector is specified (usually 3). Then, each coordinate, e.g. x y z. After the joint position, its freedoms are defined. DEGREES_OF_FREEDOM is followed by two integers. The first is the count of rotational freedoms, and the second is the count of translational freedoms. Then comes the list of freedom parameters for which there are three for each freedom. The current value is defined, then the freedom axis, then the freedom limits. Rotational limits and values are in degrees and translational ones are in meters.

```
CHILDREN 3
  1 // Neck
  2 // Upper Left Arm
  3 // Upper Right Arm
```

The list of child indices are defined here

```
ENDMESH
```

Mandatory ending of the segment information

```
ENDFIGURE
```

Mandatory ending of the figure information

7.3.3 MESH Format

The MESH format is the easiest format I have found for tetrahedral meshes. It is the default format for the free Internet program MEDIT available from http://www.ann.jussieu.fr/~frey/logiciels/medit.html. Pascal J. Frey created this format to replace the inadequacies of prior formats. I have found it necessary to add some key-words to extend the format capabilities. This format is composed of a single (binary or text) data file. Its structure is organized as a series of fields identified by keywords. The blanks, newlines or carriage returns, and tabs are considered as item separators. A comment line starts with the character # and ends at the end of the line. The comments are placed exclusively between the fields. The mesh file must start with the descriptor:

```
MeshVersionFormatted 1
Dimension 3
```

The other fields supported by MEDIT are either required or facultative. The required fields correspond to the geometry (*i.e.* the coordinates) and to the topology description (*i.e.* the mesh entities). In the following tables, the term v_i indicates a vertex number (*i.e.* the ith vertex in the vertex list), e_i is an edge number, t_i is a triangle number, T_i is a tetra-

hedron number, and q_i is a quadrilateral number. Notice that the vertices coordinates are real numbers in single precision.

Table 5 Primitives in MESH Format

Keyword	Card.	Syntax	Range
`Vertices`	np	$x_i\, y_i\, z_i\, ref_i$	$i{:}[1,np]$
`Edges`	ne	$v^1_i\, v^2_i\, ref_i$	$i{:}[1,ne],\, v^j_i{:}[1,np]$
`Triangles`	nt	$v^j_i\, ref_i$	$i{:}[1,nt],\, j{:}[1,3],$ $v^j_i{:}[1,np]$
`Quadrilaterals`	nq	$v^j_i\, ref_i$	$i{:}[1,nq],\, j{:}[1,4],$ $v^j_i{:}[1,np]$
`Tetrahedra`	$ntet$	$v^j_i\, ref_i$	$i{:}[1,ntet],\, j{:}[1,4],$ $v^j_i{:}[1,np]$
`Hexaedra`	nh	$v^j_i\, ref_i$	$i{:}[1,nh],\, j{:}[1,8],$ $v^j_i{:}[1,np]$

The description of constrained entities or singularities follows. In particular, a corner (keyword `Corner`) is a C^0 continuity point (this type of item is necessarily a mesh vertex). By analogy, a `Ridge` is an edge where there is a C^0 continuity between the adja-

cent faces. The fields of type `Requiredxx` make it possible to specify any type of entity that must be preserved by the meshing algorithm.

Table 6 Preservation Adjectives in MESH Format

Keyword	Card.	Syntax	Range
`Corners`	nc	v_i	$i{:}[1,nc]$, $v_i{:}[1,np]$
`RequiredVertices`	nrv	v_i	$i{:}[1,nrv]$, $v_i{:}[1,np]$
`Ridges`	nr	e_i	$i{:}[1,nr]$
`RequiredEdges`	nre	e_i	$i{:}[1,nre]$

As mentioned above, it is also possible to specify normals and tangents to the surface. The normals (respectively tangents) are given as a list of vectors. The normal at a vertex, keyword `NormalAtVertices`, is specified using the vertex number and the index of the corresponding normal vector. The normal at a vertex of a triangle, `NormalAtTriangleVertices`, corresponds to the combination of the triangle number, the index of the vertex in the triangle and the index of the normal vector at this vertex. The keyword `NormalAtQuadrilateralVertices` is handled similarly. The tangent vectors are described in the same way.

Table 7 Optional Adjectives of Primitives in MESH Format

Keyword	Card.	Syntax	Range
`Normals`	nn	$x_i\, y_i\, z_i$	i:[1,nn]
`Tangents`	nnt	$x_i\, y_i\, z_i$	i:[1,nnt]
`NormalAtVertices`	nv	$v_i\, n_i$	i:[1,nv], v_i:[1,np], n_i:[1,nn]
`NormalAtTriangleVertices`	ntv	$t_i\, v_i\, n_i$	i:[1,ntv], t_i:[1,nt], v_i:[1,np], n_i:[1,nn]
`NormalAtQuadrilateralVertices`	nqv	$q_i\, v_i\, n_i$	i:[1,nqv], q_i:[1,nq], v_i:[1,np], n_i:[1,nn]
`TangentAtEdges`	te	$e_i\, v_i\, t_i$	i:[1,te], e_i:[1,ne], v_i:[1,np], t_i:[1,nnt]
`*Colors`	$ncc\ p$	$r_i\ \ g_i\ \ b_i$ a_i	i:[1,ncc], p:[3,4], a_i only for p=4
`*ColorAtVertices`	ncv	$v_i\, c_i$	i:[1,ncv], v_i:[1,np], c_i:[1,ncc]
`*NeighborAtTriangleEdges`	nte	$t^1_i\, f\, t^2_i$	i:[1,nte], f:[1,3], t^j_i:[1,nt]

`*NeighborAtTetrahedronFaces`	*ntf*	$T^1_i f T^2_i$	i:[1,*ntf*], f:[1,4],
			T^j_i:[1,*ntet*]

* Keywords added for this research.

Finally, the data structure must end with the keyword: End. The new color key-words are handy for coloring the mesh. The neighbor keyword reduces the amount of time it takes to traverse a mesh. Knowing the neighbor at each face of a tetrahedron can determine which tetrahedron to go when moving through a mesh. If there is no neighbor, then the tetrahedron is on the outside.

7.3.4 PLY Format

The Polygon File Format (PLY) is another popular meshing format but only for 2D (surface) meshes. It is also known as the Stanford Triangle Format. It can be ASCII or binary and contains almost all that is needed for triangulated meshes. The format is originally from Stanford and is documented with examples and source code there (http://graphics.stanford.edu/data/3Dscanrep). This research program has incorporated the ability to read PLY files for the figure animation.

7.3.5 C3D Format

The Coordinate 3D (C3D) file format is only used for storing the raw motion cap-ture data. It is a good format and is the oldest in the business. It originally came in 1986 from Dr. Andrew Danis but is now part of a non-profit organization at http://www.c3d.org. The file format is a subset of a more general (ADTECH) format that is binary and stores both parameter information and raw data. The data can be stored in

little-endian (Intel), big-endian (MIPS, PPC), and DEC binary formats for 16-bit integers and 32-bit floats. Both analog and digital information can be stored along with their descriptions and measurement units. All of the data that comes from Carnegie Melon Graphics Lab is stored in C3D format.

7.4 User's Guide to Program

The following sections describe the use of the application that was built for the research. The application was designed using the Xcode 1.5 integrated development environment available for free from Apple and comes with the MacOSX 10.4 operating system. There are 45270 lines of code with a total McCabe's Cyclomatic Complexity Number of 8112. The application follows the recommended Macintosh human interface and is compatible with MacOSX 10.1 through 10.4.8 though only tested on 10.3 and 10.4. The engine that drives the calculations is written entirely in standard portable C++ and the graphics panes are written in OpenGL. The user interface is written in ObjectiveC that compliment the NIB files for Interface Builder. Everything except the user interface is portable. The code was compiled for the PowerPC G4 processor and was tested on a PowerBook G4.

7.4.1 Menu

The File Menu provides the basic file saving and retrieving capabilities some of which are available in the Options Pane as well. A picture of the File Menu open is in
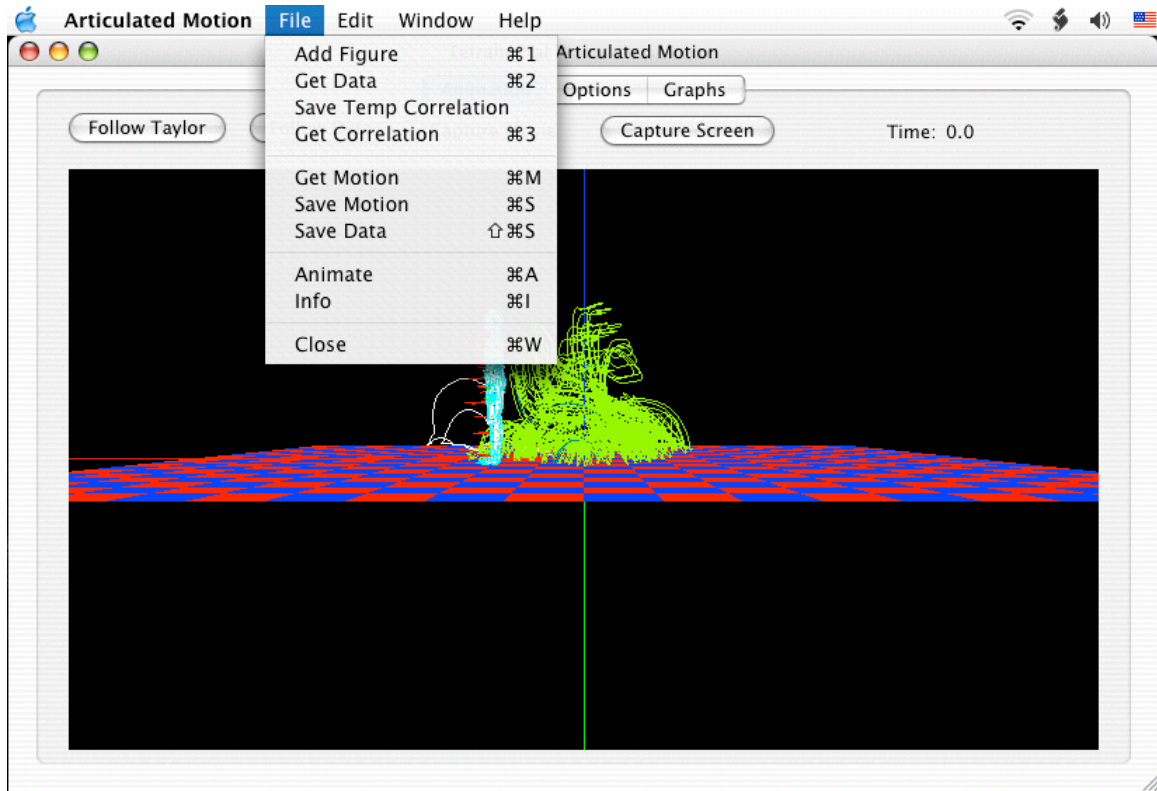
Figure 28.

Figure 28 File Menu GUI

Add Figure

The Add Figure menu item (optionally use ⌘-1) allows the user to add a figure to the graphics. A figure can be an Articulated Tetrahedral Mesh (ATM) file (as explained in Chapter 7.3.2); a MESH formatted file (both tetrahedral and triangulated); and a PLY formatted file. Only the ATM file can be used for the hierarchical analysis of the motion capture data. When this menu item is picked, a standard Open dialog is presented for choosing a file. When a file is chosen, the file is parsed and presented in the Animation Pane. The Options Pane text for the figure is filled out with the path of the file and the height of the figure is filled out.

Get Data

This menu item (optionally use -2) will present a standard Open dialog to allow the user to choose a C3D motion capture data file. The selected file will associate the data with the currently active figure (see the Options Pane). Once chosen, the file is parsed and connected lists of data frames are filled out.

Save Temp Correlation

This menu item allows the user to save a temporary correlation text file associated with the C3D data file that was previously chosen. The text file can then be edited to allow a hierarchical correlation of the data markers with segments on the figure. The number of markers is stated on the first line (e.g. 41 WANDS). The marker set file name is stated on the second line (e.g. MARKERSET vicon512.txt). Then comes a list of marker correlations (e.g. Jim:RTHI = RTHI Right thigh). The left side of the equals sign is the single word that the marker is labeled as in the data (e.g. Jim:RTHI). The right side has a single word that corresponds to a marker in the above-mentioned marker set file followed by a free-form description.

Get Correlation

This menu item (optionally use -3) presents the user with an Open dialog to retrieve a correlation file. The file is parsed and the left side is looked up in the C3D data and the right side is looked up in the marker set file. The marker set file determines which segment to attach the marker and where on the segment. If the marker is not to be used in the correlation, just put some uncorrelated name (e.g. Unknown) on the right hand side.

Get Motion

This menu item retrieves motion files that have been previous saved or manually created. Motion files allow the user to create motion of an articulated figure based on a Taylor expansion of any or all degrees of freedom.

Save Motion

This menu item will save the Taylor expansions that were created when the "Analyze" button is pressed in the Graphs Pane.

Save Data

This menu item will save the joint angles that were created when the "Analyze" button is pressed in the Graphs Pane.

Animate

This menu item will start the animation.

Info

This menu item will enable some graphics hardware information to be displayed on the Animation Pane.

Close

This menu item will close the window.

7.4.2 Options Pane

The Options plane contains all of the figure, data, and animation options available. On the right are all of the files that are used in the current analysis. The "Marker Set Dir" is the directory that all marker set files are located for the correlation. The "Figure" file is the currently active articulated model. The "Motion Data" file is the current C3D being analyzed. The "Correlate Data" file is the correlation file that binds the data with the hierarchical figure. The "replace" checkbox will replace the current figure with the opened figure if it is on. Otherwise the opened file will be added to the scene. The "ft->m" checkbox will convert the motion data from feet to meters if on. The "x2" button will multiply the sampling rate by two. The last two buttons are there for convenience since the data does not always specify the correct units or sampling rate. Once the data is read in, the height and sampling rate text can be edited to whatever size and the internal data will be updated. The same goes for the Figure information. The height of the figure can be edited.
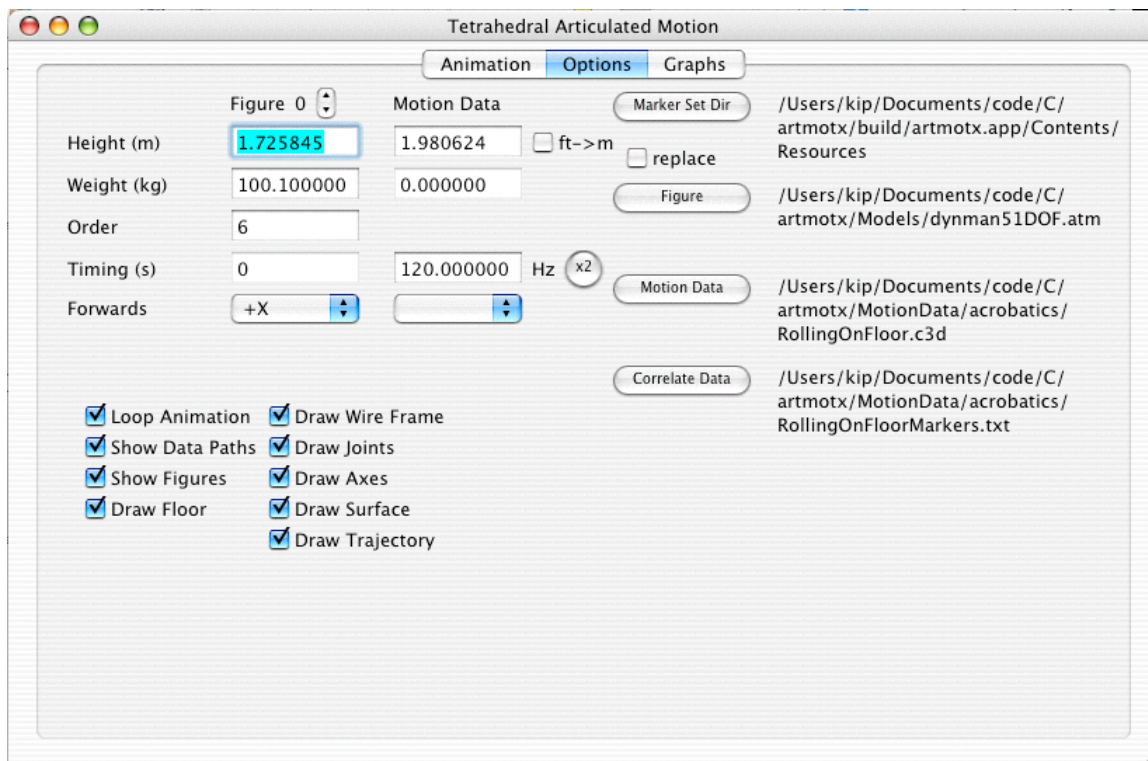
Figure 29 Options Pane GUI

Drawing Options

The many checkboxes allow the selection of various things to be drawn in the Animation Pane. "Loop Animation" will make the animation repeat when the maximum time of the data is reached. "Show Data Paths" will display the paths taken for each marker in the data. The paths will be colored lighter if the marker has been correlated to the figure. "Show Figures" will display all figures that have been read in. "Draw Floor" will ... draw the floor?! The other checkboxes draw extra little goodies on the figure.

7.4.3 Animation Pane

This pane is where all of the action happens. Select this pane once all of the files and options are set in the Options Pane. The time of the animation is displayed in the upper right in seconds. The Capture Screen button will take an exact snapshot of the

OpenGL window and allow the user to save it as a TIFF file. The TIFF file is a 32-bit ABGR8888 color file. The "Capture Movie" checkbox is similar except that it saves many TIFF files at 0.1 second intervals for the entire loop. The "Follow Data" starts the animation process that draws the moving figure and the skeleton in the data. There are options for panning the scene around. If the Command key ( ) is held down while the mouse button is held down and the screen is dragged by the mouse, then the scene is dragged up/down or left/right. If the Option key is held down with the mouse button, the user can zoom in or out of the scene with the mouse. With just the mouse button and the mouse dragging, the user can rotate the scene around the (0,0,0) point.
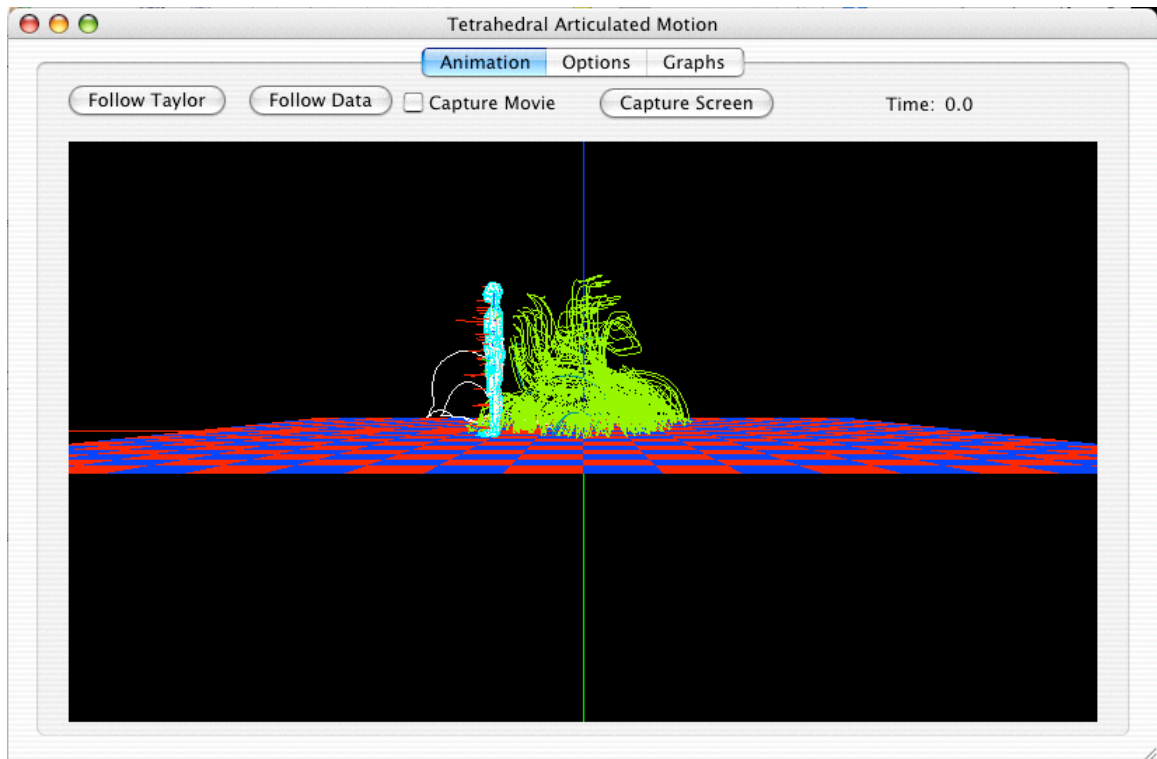
Figure 30 Animation Pane GUI

7.4.4 Graphs Pane

This pane was added for analysis of joint angles was just a visualization aid for research. The analyze button currently tries to determine the angles which are needed to rotate in order to follow the current dataset. Once the analysis is done, a joint can be picked from the pop-up. When the user slides the horizontal time bars left and right, a new Taylor expansion will be generated for the time window. Taylor expansions help in analyzing the predictive-ness of the angle paths.

Figure 31 Graphs Pane GUI

7.5 Programmer's Reference

7.5.1 Class Diagrams

The articulated figure designed for this research is for general-purpose articulated modeling. The original attempt was to use it for physical based calculations. With the advent of the Minimum Variance Method, the model has been resigned to a simple organizational tool for the hierarchical tree traversals. The entire implementation is explained here for future use. The C++ object model is set up in a hierarchical manner for the articulated figure. An articulated figure (ArtFigure.cpp) is made of segments (Segment.cpp). A segment is a rigid body (RigidBody.cpp) that is linked to another by a joint. The rigid body is a shape that can be translated and rotated, but not molded. The rigid body is made of a mesh of tetrahedrons (TetrahedralMesh.cpp). The mesh is a face-connected list of tetrahedrons. Only faces on the surface have no connected tetrahedrons and are available for drawing. A tetrahedron (Tetrahedron.cpp) is a four sided figure with four vertices assigned. Each side of the tetrahedron is a triangle (Triangle.cpp). Since adjacent tetrahedrons share vertices, the tetrahedrons will contain only references to its points. All of the points for the tetrahedral mesh will be stored in a linear array inside the TetrahedralMesh instance. This allows easy access to the points for quick translations and rotations without duplication of effort. Figure 32 is a UML Diagram describing the relationships between objects.
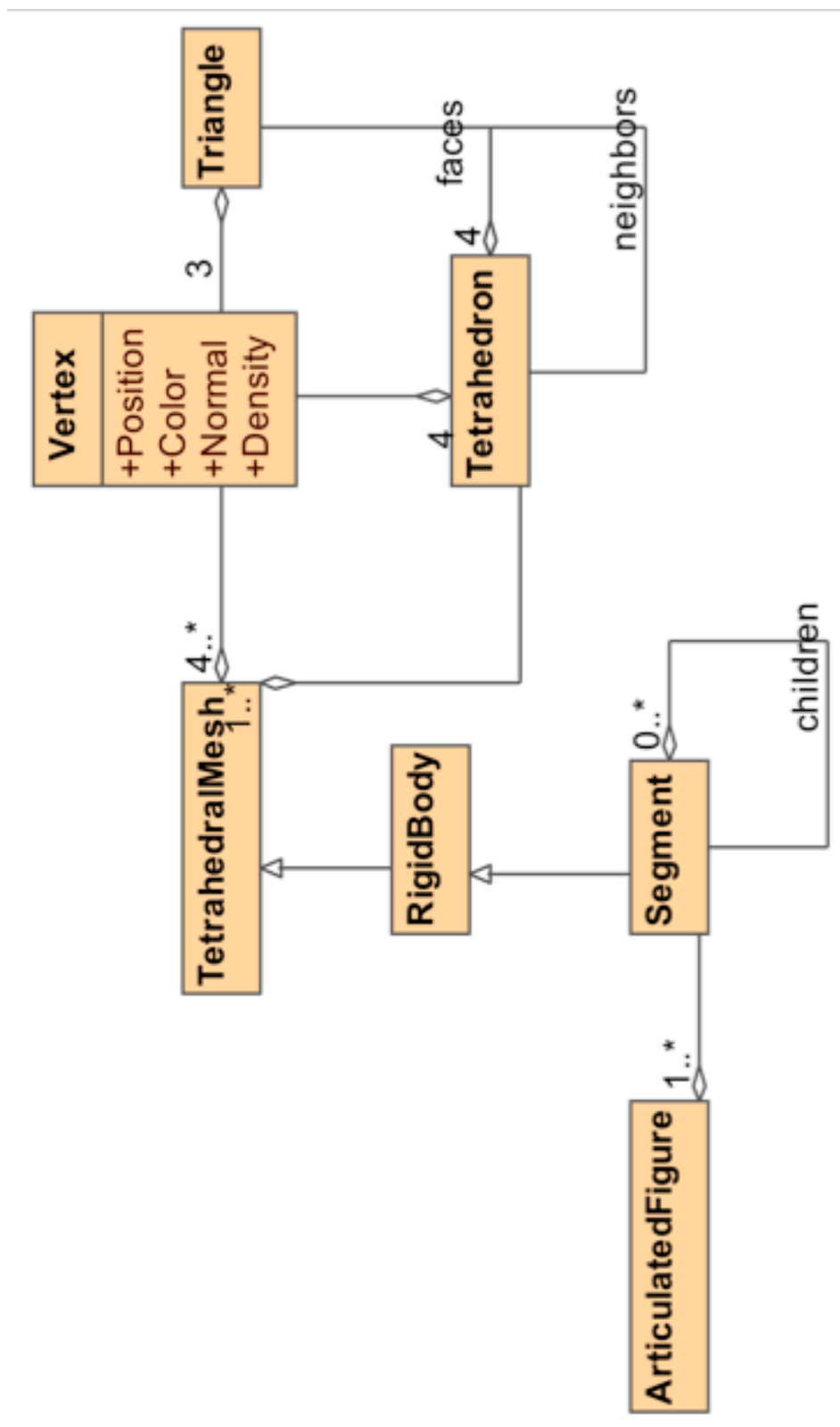
Figure 32 UML Diagram of Articulated Figure

7.6 C++ Implementations

This section contains the crucial implementations for the novel algorithms in this thesis. It starts with the implementation of the UGDK, followed by IGDK. The next algorithm involves the collecting of the raw data and turning them into relative positions. The calculation of the rotation point follows.

7.6.1 Unbiased Generalized Delogne-Kása Method

This snippet of C++ code implements the core of the Unbiased Generalized Delogne-Kása Method. Some of the standard linear algebra operations are hidden inside of the Vector and Matrix classes.

```cpp
void VectorDataSet::init( const Vector * v,
                          unsigned int n,
                          double eps )
{
  unsigned int i;
  Vector d;
  double d2;

  if( n == 0 )
  {
    cerr << "ERROR: VectorDataSet::init"
         << " no data to init set"
         << endl;
    return;
  }
  init( v->numElements );
  for( i=0; i<n; i++ )
  {
    moment1 += v[i];
  }
  moment1 /= n;
  for( i=0; i<n; i++ )
  {
    d = v[i]-moment1;
```

```
      covariance += d.SquareTranspose();
      d2 = d*d;
      moment3 += d*d2;
      moment2 += d2;
    }
    moment2 /= n;
    if( n > 1 )
    {
      covariance /= n-1;
      moment3 /= n-1;
      covariance -= eps*eps; // compensate for bias
    }
    else
    {
      covariance = 0.0;
      moment3 = 0.0;
    }
    Matrix decomp = covariance.CholeskyDecomposition();
    covarianceInverse = decomp.CholeskyInverse();
    center = moment1 + 0.5*decomp.CholeskyBacksubstitution(moment3);
    centerDiff = 1.0e20;
    difMoment1 = 1.0e20;
    radius = sqrt(((n-1.0)/n)moment2+(center-moment1)*(center-moment1));
    numData = n;
    numCenterData = 1;
    centerAvg = center;
}
```

7.6.2 Incrementally Improved Generalized Delogne-Kása

This code snippet adds a vector to the data set and improves the various values
that were previously calculated in Section 7.6.1.

```
void VectorDataSet::operator+=( const Vector& a )
{
  unsigned int i;
  Vector b,d;
  double d2,q1,qn1,q2,qp1;
  Matrix m;
  if( numData > 0 )
  {
    difMoment1 += a - lastValue;
  }
  lastValue = a;
```

```
q1 = (numData/(numData+1.0)); // < 1
qn1 = (numData-1.0)/numData;  // < 1
q2 = (numData+2.0)/numData;   // > 1
qp1 = (numData+1.0)/numData;  // > 1
difMoment1 = (a-moment1)/(numData+1.0);
moment1 += difMoment1; // average
d = a-moment1;
d2 = d*d
b = covarianceInverse*d;
moment2 += (qp1*d2 - moment2)/(numData+1.0);
covarianceInverse += (covarianceInverse-(numData/((numData-
      1.0)*q1+(d*b)))*b.SquareTranspose())/(numData-1.0);
covariance += (qp1*d.SquareTranspose() - covariance)/numData;
centerDiff = (covarianceInverse*d)*(d*(a-center) -
      0.5*(moment2+d2))/(numData+1.0);
center += centerDiff;
centerAvg += (center-centerAvg)/(numCenterData+1.0);
moment3 += ((-2.0*covariance-moment2*qp1+d2*(qp1*q2))*d-
      moment3)/numData;
radius = sqrt(qn1*moment2+(center-moment1)*(center-moment1));

for( i=0; i<dimension; i++ )
{
  if( a[i] < minimum[i] )
  {
    minimum[i] = a[i];
  }
  if( a[i] > maximum[i] )
  {
    maximum[i] = a[i];
  }
}
numData++;
numCenterData++;
}
```

7.6.3 Collecting the Raw Data for Rotation Point

This function collects the raw data and creates an array of positions relative to the parent segment. The inputs are the time window beginning and end. The outputs are the collection of relative points and whether the function succeeded.

```
bool Segment::CollectRelativeDataPoints( Units::second t1,
                                         Units::second t2 )
{
```

```cpp
unsigned int i,k,n=1,n1=n;
double maxLen = 0;
PositionVector dataCenter2;
Matrix dataAxes2;
Units::second t3 = 0,deltaT,lastTime;
EventFrame::ElementType evl[1] = {NULL};
Event * e[n];
EventFrame events;
unsigned int best = 0;
Vector pi(3);

// the parent must exist and have data
if( numCorrelated == 0 ||
    parent == NULL )
{
  return false;
}
// find the farthest data point
for( i=0; i<correlatedFrame.numElements; i++ )
{
  for( k=i+1; k<correlatedFrame.numElements; k++ )
  {
    if( correlatedFrame.lengths[i][k] > maxLen )
    {
      best = i;
      maxLen = correlatedFrame.lengths[i][k];
    }
  }
}
evl[0] = correlatedFrame[best];
if( evl[0] == NULL )
{
  return false;
}
events.setEvents(evl,n);
deltaT = events.minTimeBetweenEvents;
if( t1 < events.minTime )
{
  t1 = events.minTime;
}
if( t2 > events.maxTime )
{
  t2 = events.maxTime;
}
t3 = t1 - deltaT; // so first frame is retrieved
pathToFollow.RemoveAll();
while( events.NextAbsoluteStableFrame(n1,e,t3) && (t3 <= t2) )
{
```

```cpp
    if( !parent->GetSameDataAxes( t3, dataAxes2, dataCenter2 ) )
    {
      // bad frame, continue to next
      continue;
    }
    for( i=0; i<n; i++ )
    {
      if( e[i] != NULL )
      {
        pi = (e[i]->position - dataCenter2)*dataAxes2;
        pathToFollow.InsertEvent(pi,
                                 e[i]->time,
                                 e[i]->accuracy,
                                 e[i]->time_accuracy);
      }
    }
  }
  return true;
}
```

7.6.4 Rotation Point Calculation of Segment

This function takes the collection of relative data points and calculates the relative

rotation point in the data. It also calculates the best fit planar normal of the data which is

needed for marker sets of one marker on a segment. It returns the relative rotation point.

```cpp
PositionVector Segment::GetDataRotationPoint()
{
  Event * e = NULL;
  unsigned int j;
  Vector p(3),pi(3),p3(3),mean(3),pip(3),pir(3);
  Vector b,r;
  Matrix A(3,3),P2(3,3),a,Ainv;
  //Vector::ElementType p2=0.0,q,L=0.0;
  double err(0);
  static const int N = 100000;
  int n;
  VectorDataSet dataSet;
  Vector vs[N];

  kalmanFilterFailed = true;
  if( CollectRelativeDataPoints() )
  {
    n = 0;
```

```
      e = pathToFollow.beginning;
      while( e && n < N )
      {
        vs[n] = e->position;
        e = e->next;
        n++;
      }
      dataSet.init(vs,n,pathToFollow.beginning->accuracy);
      while( e && !dataSet.sphereHasConverged(0.01) )
      {
        dataSet += e->position;
        e = e->next;
        n++;
      }
      if( n > 0 )
      {
        j = n;
        r = dataSet.center;
        double cn = dataSet.covariance.ConditionNumber();
        Vector v = dataSet.covariance.NullSpace(1.000001/cn).Column(0);
        relativeDataParentOneDOFaxis = v;
        Vector r1 = r + v*((p-r)*v);

        if( cn > 10000.0 ) // project onto plane
        {
          r = r1;
        }
      }
    }
  return PositionVector(r);
}
```

7.6.5 Constants Calculation of Hierarchical Articulated Data

This function calculates the constants necessary to draw the skeleton at any time frame. There are no inputs except for the available raw data. The outputs are the constants that are set for each segment. This function must be called from the root segment or the hierarchy will break.

```
void Segment::SetDataRelativeStuff() // call from root
{
  Matrix dataAxes = Matrix::Identity(3);
  PositionVector dataCenter,rot,c,v;
```

```cpp
unsigned int i;
Units::second t = 0;
Event * e[numCorrelated];

unsigned int n = 0,n1=numCorrelated;
if( correlatedFrame.NextFrame(n1,e,t) )
{
  c = 0.0;
  for( i=0; i<numCorrelated; i++ )
  {
    if( e[i] != NULL )
    {
      c += e[i]->position;
      n++;
    }
  }
  if( n > 0 )
  {
    c /= n;
  }
}
if( parent )
{
  // calculate rotation point from available data
  relativeDataParentRotationPoint = GetDataRotationPoint();
  if( parent->numCorrelated == 0 ) // set previous segment's
  {
    parent->relativeDataParentRotationPoint =
            relativeDataParentRotationPoint;
  }
  t = 0.0;
  if( parent->GetDataAxes(t,dataAxes,dataCenter) )
  {
    rot = dataCenter + dataAxes*relativeDataParentRotationPoint;
    v = dataAxes*relativeDataParentOneDOFaxis;
    relativeDataJointAxesZero =
                dataAxes.Transpose()*Matrix::Identity(3);
  }
  else
  {
    cerr << "No parent data axes for " << name << endl;
    rot = PositionVector(0.0,0.0,0.0);
    relativeDataJointAxesZero = Matrix::Identity(3);
  }
  t = 0.0;
  // set segment's relative stuff to markers
  if( GetDataAxes(t,dataAxes,dataCenter) )
  {
```

```cpp
      relativeDataBodyAxes = dataAxes.Transpose()*Matrix::Identity(3);
      relativeDataJointAxes = dataAxes.Transpose()*Matrix::Identity(3);
      relativeDataRotationPoint = (rot - dataCenter)*dataAxes;
      relativeDataCentroid = (c - dataCenter)*dataAxes;
      relativeDataOneDOFaxis = dataAxes.Transpose()*v;
    }
    else
    {
      cerr << "No self data axes for " << name << endl;
    }
  }
  else // root
  {
    // no rotation point so just get the centroid of data
    t = 0.0;
    if( !GetDataAxes(t,dataAxes,dataCenter) )
    {
      cerr << "Damn, root really needs >= 3 correlated markers"
           << endl;
    }
    relativeDataCentroid = (c - dataCenter)*dataAxes;
    relativeDataRotationPoint = relativeDataCentroid;
    relativeDataOneDOFaxis = DirectionVector(1.0,0.0,0.0);
    relativeDataParentOneDOFaxis = DirectionVector(1.0,0.0,0.0);
    relativeDataParentRotationPoint = PositionVector(0.0,0.0,0.0);
    relativeDataJointAxes = dataAxes.Transpose()*Matrix::Identity(3);
    relativeDataJointAxesZero=dataAxes.Transpose()*Matrix::Identity(3);
    relativeDataBodyAxes = dataAxes.Transpose()*Matrix::Identity(3);
  }
  //cout << "DataGrouping " << CheckDataGrouping() << endl;
  for( i=0; i<numChildren; i++ )
  {
    children[i]->SetDataRelativeStuff(); // recursive
  }
}
```

7.6.6 Calculation of fixed axes of data

This function calculates the center and the three axes for a segment based upon the previously calculated constants and the current raw data. The input t is the time at which to extract the data. The function outputs the axes in the form of a 3x3 matrix and

the center in absolute coordinates if successful. If not successful, the function returns false.

```cpp
bool Segment::GetDataAxes( Units::second& t,
                           Matrix& dataAxes,
                           PositionVector& dataCenter ) const
{
  Event * e[3] = {NULL,NULL,NULL};
  Units::second oldT = t;

  // a segment without axes, use parent's
  if( parent && axesFrame.numElements == 0 )
  {
    if( !parent->GetDataAxes( t, dataAxes, dataCenter ) ) // recursive
    {
      return false;
    }
  }
  // root and all segments with many points gets in here
  else if( axesFrame.numElements == 3 )
  {
    unsigned int n = 3;
    if( !axesFrame.NextAbsoluteFrame(n,e,t) )
    {
      if( t < axesFrame.maxTime )
      {
        cerr << "Missing events for " << jointName
             << " t " << t << endl;
      }
      return false;
    }
    dataCenter = e[0]->position;
    dataAxes = Matrix::Axes(dataCenter,e[1]->position,e[2]->position);
  }
  else if( !parent ) // segment must have parent beyond this point
  {
    return false;
  }
  else if( axesFrame.numElements == 2 ) // need parent's stuff set
  {
    unsigned int n = 2;
    Matrix pDataAxes;
    PositionVector pDataCenter;
    if( !axesFrame.NextAbsoluteFrame(n,e,t) )
    {
      return false;
```

```
    }
    // recursive
    if( !parent->GetSameDataAxes( t, pDataAxes, pDataCenter ) )
    {
      return false;
    }
    dataCenter = pDataCenter+pDataAxes*relativeDataParentRotationPoint;
    dataAxes = Matrix::Axes(dataCenter,e[0]->position,e[1]->position);
  }
  else if( axesFrame.numElements == 1 ) // need parent's stuff set
  {
    unsigned int n = 1;
    Matrix pDataAxes;
    PositionVector pDataCenter,r2;

    if( !axesFrame.NextAbsoluteFrame(n,e,t) )
    {
      return false;
    }
    // recursive
    if( !parent->GetSameDataAxes( t, pDataAxes, pDataCenter ) )
    {
      return false;
    }
    dataCenter = pDataCenter+pDataAxes*relativeDataParentRotationPoint;
    r2 = dataCenter + pDataAxes*relativeDataParentOneDOFaxis;
    dataAxes = Matrix::Axes(dataCenter,r2,e[0]->position);
  }
  else
  {
    return false;
  }
  return true;
}
```

7.6.7 Drawing Rotation Points with Constants of Motion

This function is called after the rotation points have been calculated and the hierarchical skeleton has been defined. It retrieves the current raw 3D positional data and calculates where the rotation points and axes are supposed to be based on the previous analysis. The radius input is for drawing a sphere around the current raw data. The t in-

put is the time at which the data is retrieved. There are no results except those that are drawn on the OpenGL window.

```cpp
void Segment::DrawNearestEvents( double radius,
                                 Units::second t )
{
  unsigned int i;
  Matrix dataAxes,dataAxes0,dataAxes1;
  PositionVector dataCenter,dataCenter0,dataCenter1,rot,rot1,pos,ep;

  if( t<lastTime )
  {
    count = 0;
  }
  lastTime = t;

  Color::Green.glColor();
  correlatedFrame.Draw(radius*0.5,t); // draw dots for data
  if( parent && parent->GetDataAxes( t, dataAxes0, dataCenter0 ) )
  {
    rot = dataCenter0 + dataAxes0*relativeDataParentRotationPoint;
  }
  else if( parent != NULL )
  {
    return;
  }
  t -= 0.01;
  if( GetDataAxes( t, dataAxes, dataCenter ) )
  {
    if( parent == NULL )
    {
      rot = dataCenter + dataAxes*relativeDataCentroid;
    }
  }
  else
  {
    return;
  }
  Color::White.glColor();
  glBegin(GL_LINES);
  for( i=0; i<numChildren; i++ )
  {
    if( children[i]->numCorrelated ||
        children[i]->numChildren > 0 )
    {
      rot.glVertex();
```

```
        rot1 = datacenter + dataAxes*children[i]->
                                relativeDataParentRotationPoint;
        rot1.glVertex();
    }
  }
  if( numCorrelated > 0 &&
      numChildren == 0 )
  {
    rot.glVertex();
    (dataCenter + dataAxes*relativeDataCentroid).glVertex();
  }
  glEnd();
  if( numChildren == 0 ) // end effector
  {
    correlatedFrame.DrawLines(t);
  }
  count++;
}
```