Improve motion capture from RGB-D sensors for online, unilateral teleoperation of humanoid robots

Hilario Tomé

Facultat d’informatic de Barcelona
Universitat Politècnica de Catalunya

Pal Robotics S.L

June, 2012

A degree project submitted in partial fulfilment of the requirements for the degree of Computer Engineering.

Director: Adolfo Rodríguez Tsouroukdissian
Codirector: Juan Andrade-Cetto
Supervisor: Joan Aranda López
## Contents

1. **Abstract**\hfill 4  
2. **Attitude representation**\hfill 5  
   2.1 Discrete cosine matrix\hfill 5  
   2.2 Euler angles & Roll-Pitch-Yaw angles\hfill 5  
   2.2.1 Small angle approximation\hfill 6  
   2.3 Angle-Axis representation\hfill 7  
   2.4 Quaternion\hfill 8  
   2.4.1 Composition rule for quaternions\hfill 8  
   2.4.2 Small angle approximation for quaternions\hfill 9  
   2.4.3 Quaternion evolution with time\hfill 10  
   2.5 Modified Rodrigues\hfill 12  
   2.5.1 Modified Rodrigues composition\hfill 12  
   2.5.2 Modified Rodrigues evolution with time\hfill 12  
3. **Optimal state estimation**\hfill 14  
   3.1 Transformation of vector random variables\hfill 14  
   3.2 Maximum likelihood estimation\hfill 15  
   3.3 SAM (Simultaneous smoothing and mapping) example\hfill 16  
   3.4 Recursive least squares\hfill 20  
   3.5 Kalman filter\hfill 21  
   3.5.1 Kalman filter example\hfill 23  
4. **Orientation estimation and filtering**\hfill 25  
   4.1 Extended Kalman Filter\hfill 25  
   4.2 Filter implementation and results\hfill 26  
   4.2.1 Simulation\hfill 27  
5. **Human skeleton estimation and robot teleoperation**\hfill 28  
   5.1 Denavit-Hartenberg\hfill 29  
   5.2 Uncented Kalman filter\hfill 31  
   5.3 Filter implementation and results\hfill 35  
   5.4 Teleoperation\hfill 35  
6. **Attitude heading and reference system & robot head and wrist control**\hfill 37  
   6.1 Indirect complementary filter\hfill 37  
   6.2 Sensor error model\hfill 39  
   6.2.1 Gyroscope\hfill 39  
   6.2.2 Accelerometer\hfill 41
6.2.3 Magnetometer ................................................. 42
6.3 Error quaternion ................................................ 43
6.4 Measurement model ........................................... 45
   6.4.1 Accelerometer measurement model ....................... 46
   6.4.2 Magnetometer measuring model ........................... 46
6.5 Initialization .................................................... 47
6.6 Filter implementation and results ............................. 47
   6.6.1 Simulation .................................................. 48
   6.6.2 Real data ................................................... 49
6.7 Robot wrist teleoperation ...................................... 49

7 Conclusion .......................................................... 55
   7.1 Time planification ............................................. 55
   7.2 Costs & Resources ............................................ 55
1 Abstract

The first goal of this thesis is to develop a solution to teleoperate a high gain position controlled humanoid robot. The second goal is to develop a solution that tracks the humans wrist and head orientation in order add functionality to available open source skeleton trackers that lack this feature. The final goal is to also add wrist and head orientation to the teleoperation of the robot.

To solve each of this problems we will first describe in chapter one how to parametrize them. The different ways to represent the attitude of a system will be explained including their advantages and drawbacks. The one that we are mostly going to use is the quaternion, that is why we will go a bit more in depth about it.

Chapter two will explain the concept of optimal filtering and the tool that we are mostly going to use called the Kalman filter. We will show how it is derived from recursive least squares. With a simple example its shown the importance of optimal filtering.

Chapter three will develop tree applications based on the theory introduced in the previous chapters that solve the problems of this thesis. The first application is an optimal orientation and position filter. It uses the Rodrigues parametrization to filter the rotational part. This filter smooths and filters the signals of the skeleton tracker that are forwarded to the teleoperation pipeline, giving a smooth reference signal for the robot to follow.

The second application is an alternative to solving the first problem. It is a skeleton filter and tracker that is used to jointly estimate and filter simultaneously the complete state of the human. A smooth trajectory of the human joints is estimated and using forward kinematics the Cartesian positions of the human body parts are recovered in order to forward them to the teleoperation pipeline. We will also explain how the filtered skeleton can be used to the teleoperate the robot.

The last application involves adding wrist and head teleoperation to the robot. This is done designing an inertial measuring unit that tracks the orientation of the limb that is attached to. Using them in the users head and hands we can map the limbs orientation in the robot giving the robot the maximum amount of expressiveness.

In each of the applications a modification to the Kalman filter was made that allowed to solve each of the problems better. The derived filters are the extended Kalman filter, the unscented Kalman filter and the indirect complementary Kalman filter. In each application they will be introduced and derived.
2 Attitude representation

In this chapter we are going to describe how to represent the position and orientation of a body with respect to a reference frame. An example could be the measured human head position and orientation with respect to the camera reference frame. Because we can have different reference frames on which to represent a measurement we will also explain how to transform between them. We are interested in Cartesian coordinate systems, in which the position of a body can be described by a three dimensional vector in $\mathbb{R}^3$ which represents a translation from the origin of the frame and a three dimensional rotation in $SO(3)$ which represents the attitude of the body with respect to the reference frame. A rotation allows different parametrizations, each of them with different properties. We will also explain how the different parametrization affect the quality of an attitude estimate.

2.1 Discrete cosine matrix

The most common representations of a rotation is a 3x3 matrix. This matrix $R$, is often called the discrete cosine matrix. Each of its components $\cos(x'x)$, represents the cosine of the angle between the fixed frame coordinate $x$ and the moving frame coordinate $x'$ using a $3\times3$ matrix. If we want to concatenate successive rotations we only need to multiply their corresponding matrices. Because the DCM has nine components but only three degrees of freedom and the additional restriction of being an orthonormal matrix is made, giving a rotation matrix the property

$$R^TR = R R^T = I_{3\times3}$$  \hspace{1cm} (2.1)

$$R = \begin{bmatrix} \cos(x'x) & \cos(y'y) & \cos(z'z) \\ \cos(xy') & \cos(yy') & \cos(z'y') \\ \cos(xz') & \cos(yz') & \cos(z'z') \end{bmatrix}$$  \hspace{1cm} (2.2)

2.2 Euler angles & Roll-Pitch-Yaw angles

The euler angles & roll-pitch-angles are a minimum representation of a rotation in the sense that they only need tree parameters. Given tree parameters (angles) $[\phi, \theta, \psi]$ we can describe any arbitrary rotation in space. The euler angles represent
a rotation by tree consecutive rotations around the moving frame while roll-pitch-yaw angles represent it by tree consecutive rotations around the axis of the fixed frame.

Each angle rotation can be transformed into an elementary rotation matrix, giving the same composition rule as the DCM. In the inverse problem, getting the angles from a rotation matrix can be problematic due to singularities in the representation. The same singularities arise if the the angles are integrated directly. The resulting rotation matrix that comes out \([\phi, \theta, \psi]\) euler angles is

\[
\begin{align*}
R_x(\phi) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & \sin \theta \\ 0 & -\sin \theta & \cos \theta \end{bmatrix} \\
R_y(\theta) &= \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \\
R_z(\psi) &= \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{align*}
\]

which concatenated as \(R_x(\phi)R_y(\theta)R_z(\psi)\) gives the following matrix

\[
R = \begin{bmatrix}
\cos \theta \cos \psi & \cos \phi \sin \psi + \sin \phi \sin \theta \cos \psi & \sin \phi \sin \psi - \cos \phi \sin \theta \cos \psi \\
-\cos \theta \sin \psi & \cos \phi \cos \psi - \sin \phi \sin \theta \sin \psi & \sin \phi \cos \psi + \cos \phi \sin \theta \sin \psi \\
\sin \theta & -\sin \phi \cos \theta & \cos \phi \cos \theta
\end{bmatrix}
\]

\[\text{(2.4)}\]

\[\text{2.2.1 Small angle aproximation}\]

Matrix \((2.4)\) can be simplified if the rotation angles are small, this is do to the fact that

\[
\begin{align*}
\sin \theta & \approx \theta \\
\cos \theta & \approx 1
\end{align*}
\]

\[\text{(2.5)}\]

Using this result the rotation matrix \((2.4)\) can be expressed as infinitesimal rotation matrix composed of three infinitesimal angles \([\delta \theta_1 \delta \theta_2 \delta \theta_3]\)
\[ \mathbf{R}_z(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \delta \theta_1 \\ 0 & -\delta \theta_1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\delta \theta_2 \\ 0 & 1 & 0 \\ -\delta \theta_2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \delta \theta_3 & 0 \\ -\delta \theta_3 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

\[ = \begin{bmatrix} 1 & \delta \theta_3 & -\delta \theta_2 \\ -\delta \theta_3 & 1 & \delta \theta_1 \\ \delta \theta_2 & -\delta \theta_1 & 1 \end{bmatrix} \]

\[ = (\mathbf{I} - \delta \Theta) \quad (2.6) \]

Where \(\delta \Theta\) is a skew symmetric matrix that represents \([\delta \theta \times]\) as defined in (2.13)

### 2.3 Angle-Axis representation

An alternative to the use of rotation matrices is the angle axis representation. Any relative orientation of coordinate frames can be uniquely determined by a rotation angle \(\Phi\) and a fixed axis through the common origin. We can derive this quantities from a rotation matrix analysing its eigenvalues and eigenvectors.

\[ \mathbf{R}_c = \lambda \mathbf{c} \]

\[ (\mathbf{R}_c)^H \mathbf{R}_c = \lambda \lambda^H \mathbf{c} \]

\[ (\lambda \lambda - 1) \mathbf{c}^H \mathbf{c} = 0 \]

\[ \lambda \lambda = 1 \quad (2.7) \]

Equation (2.7) shows that all the eigenvalues of a rotation matrix have unit norm, which makes sense since a rotation matrix does not change the magnitude of a vector when it multiplies it because is orthonormal. The eigenvalues are \(\lambda_1 = 1\), whose associated eigenvector \(\mathbf{c}_1\) will be the axis of rotation, and \(\lambda_{2,3} = e^{i\beta}\) which corresponds a rotation by an angle \(\beta\) of their eigenvectors \(\mathbf{c}_{2,3}\) around \(\mathbf{c}_1\). This angle must be the rotation of the coordinate frame around the axis, therefore \(\Phi = \beta\). Knowing that the trace of a matrix is equal to the sum of its eigenvalues we can find the value of \(\Phi\) as follows

\[ \text{trace} \mathbf{C} = \lambda_1 + \lambda_2 + \lambda_3 = 1 + e^{i\Phi} + e^{-i\Phi} = 1 + 2 \cos \Phi \]

\[ \cos(\Phi) = \frac{1}{2} (\text{trace} \mathbf{C} - 1) \quad (2.8) \]
2.4 Quaternion

The quaternion arises from the angle-axis representation. It is a parametrization without singularities and only four parameters. A quaternion is composed of four mutually dependent parameters \(q_1, q_2, q_3, q_4\), the first three components are called the vector part and the fourth component is called the scalar part. The derivation is as follows

\[
q_i = e_i \sin \left( \frac{\Phi}{2} \right) (i = 1, 2, 3)
q_4 = \cos \left( \frac{\Phi}{2} \right)
\]

(2.9)

It can be seen that the quaternion lies on a four-dimensional hypersphere as it satisfies the constraint \(q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1\). This constraint implies that only three of the parameters are independent.

The quaternion can be expressed in matrix form by substituting (2.9) into the eulers formula leading to

\[
C(q) = (q_4^2 - q^\top q)I_{3x3} + 2qq^\top - 2q_4S(q)
\]

(2.10)

where

\[
S(q) = \begin{pmatrix}
0 & -q_3 & q_2 \\
q_3 & 0 & -q_1 \\
-q_2 & q_1 & 0
\end{pmatrix}
\]

(2.11)

2.4.1 Composition rule for quaternions

Just like rotations parametrized by matrices can be combined by multiplying them, quaternions also have a composition rule that allows to combine them. The multiplication is defined as:

\[
q \otimes p = (q_4 + q_1i + q_2j + q_3k)(p_4 + p_1i + p_2j + p_3k)

= q_4p_4 - q_1p_1 - q_2p_2 - q_3p_3 + (q_4p_1 + q_1p_4 - q_2p_3 + q_3p_2)i

+ (q_4p_2 + q_2p_4 - q_3p_1 + q_1p_3)j + (q_4p_3 + q_3p_4 - q_1p_2 + q_2p_1)k

= \begin{pmatrix}
q_4p_1 + q_3p_2 - q_2p_3 + q_1p_4 \\
-q_3p_1 + q_4p_2 + q_1p_3 + q_2p_4 \\
q_2p_1 - q_1p_2 + q_4p_3 + q_3p_4 \\
-q_1p_1 - q_2p_2 - q_3p_3 + q_4p_4
\end{pmatrix}
\]

(2.12)

The quaternion multiplication (2.12) can also be written in matrix form. We will first explain how to write a cross-product in matrix form using a skew-symmetric matrix defined as \([q \times]\)
\[
[q \times] = \begin{bmatrix}
0 & -q_3 & q_2 \\
q_3 & 0 & -q_1 \\
-q_2 & q_1 & 0
\end{bmatrix}
\]  
(2.13)

Using (2.13) we can write a cross product as

\[
q \times p = \begin{bmatrix}
q_2p_3 - q_3p_2 \\
q_3p_1 - q_1p_3 \\
q_1p_2 - q_2p_1
\end{bmatrix} = \begin{bmatrix}
0 & -q_3 & q_2 \\
q_3 & 0 & -q_1 \\
-q_2 & q_1 & 0
\end{bmatrix} \begin{bmatrix}
p_1 \\
p_2 \\
p_3
\end{bmatrix} = q \times p
\]  
(2.14)

The quaternion multiplication can now be rewritten in matrix form as

\[
q \otimes p = \begin{bmatrix}
q_4 & q_3 & -q_2 & q_1 \\
-q_3 & q_4 & q_1 & q_2 \\
q_2 & -q_1 & q_4 & q_3 \\
-q_1 & -q_2 & -q_3 & q_4
\end{bmatrix} \begin{bmatrix}
p_1 \\
p_2 \\
p_3 \\
p_4
\end{bmatrix}
\]  
(2.15)

which can be written in a compact form as

\[
q \otimes p = \mathcal{L}(q)p
\]  
(2.16)

It is more efficient than the combined rotation matrices since it only needs 16 multiplications instead of 27.

2.4.2 Small angle approximation for quaternions

Using the fact that the cosine and sine of an small angle can be simplified to (2.5), it allows us to parametrize a quaternion that represents a small rotation in another way by using only three parameters.

We will use the small angle approximation of the quaternion when we want to describe the error between the computed orientation between two frames and its real value. With an error vector there is a covariance matrix associated and since the quaternion has the unit norm constraint it makes this matrix singular. We avoid the problem of having a degenerate covariance 4x4 matrix if we use this new small angle representation (a 4x4 singular matrix now becomes a full rank 3x3 valid covariance matrix). For \( \Phi = \delta \theta \) derivation is as follows
Given that we have a local coordinate frame $L$ that moves with respect to time to the fixed frame $G$, we can compute the time derivative of the quaternion that represents the rotation between the two coordinate frames. The derivative is found computing the limit of the following difference

$$
\frac{L_G(t)}{\Delta t} \mathbf{q}(t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \left( L_G(t+\Delta t) \mathbf{q} - L_G(t) \mathbf{q} \right)
$$

(2.19)

We are going to use the fact that $L_G(t+\Delta t) \mathbf{q}$ can be written as the product of two quaternions: the first one that represents a small rotation in time with respect to the local frame and a second quaternion that represents a rotation from the global frame to the local frame $L(t) \otimes \mathbf{q}$. The derivation is as follows: first we re-write the quaternion incremented $\Delta t$ in time and also multiple the original
quaternion by the identity quaternion. In this way we can factor out the original quaternion and use the small quaternion approximation defined in (2.18). Lastly, we use (2.15) to express the result in matrix form

\[
\frac{L(t)}{G} q(t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \left( \frac{L(t + \Delta t)}{L(t)} q - \frac{L(t)}{G} q \right) \tag{2.20}
\]

\[
= \lim_{\Delta t \to 0} \frac{1}{\Delta t} \left( L(t + \Delta t) q \otimes \frac{L(t)}{G} q - q \otimes \frac{L(t)}{G} q \right) \tag{2.21}
\]

\[
\approx \lim_{\Delta t \to 0} \frac{1}{\Delta t} \left( \begin{bmatrix} \frac{1}{2} \cdot \delta \theta & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) \otimes \frac{L(t)}{G} q \tag{2.22}
\]

\[
= \frac{1}{2} \left[ \begin{bmatrix} \omega \\ 0 \end{bmatrix} \otimes \frac{L(t)}{G} q \tag{2.23}
\right.
\]

\[
= \frac{1}{2} \left[ \begin{bmatrix} -[\omega \times] & \omega \\ -\omega^T & 0_{3x3} \end{bmatrix} \right] \frac{L(t)}{G} q \tag{2.24}
\]

\[
= \frac{1}{2} \Omega(\omega) \frac{L(t)}{G} q \tag{2.25}
\]

with \( \Omega(\omega) \) defined as

\[
\Omega(\omega) = \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & -\omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & \omega_y & -\omega_z & 0 \end{bmatrix} \tag{2.26}
\]

Since (2.25) is linear in \( q \) we can solve the system analytically using matrix exponentiation. Just as \( e^{at} \) is often the case the solution to scalar homogeneous first order differential equations, the same thing applies to systems of first order differential equations. Not to be confused, \( e^{At} \) is not the exponentiation of the individual components of the matrix, but the matrix given by the power series

\[
e^{x} = \sum_{k=0}^{\infty} \frac{1}{k!} x^k. \tag{2.27}
\]

We can apply this to solve the quaternion differential equation to get the solution

\[
\frac{L(t)}{G} q(t_{k+1}) = \frac{1}{2} \Theta(t_{k+1}, t_{k}) \frac{L(t)}{G} q(t_k) \tag{2.28}
\]

where

\[
\Theta(\Delta t) = I_{4x4} + \frac{\Delta t}{2} \Omega(w) \tag{2.29}
\]
2.5 Modified Rodrigues

We are now ready to derive the Rodrigues parametrization [3] (also called Gibbs Vector) from the quaternion. We are going to get a three value parametrization \( \mathbf{p} = (p_1, p_2, p_3) \) from the quaternion \( (q, q_4) \). Each component is derived as

\[
\rho = \frac{q}{q_4} \tag{2.30}
\]

It can be seen that now we have a singularity whenever \( q_4 \) is 0 or, in other words, when the rotation about the principal axis is equal to 180. That is why we will use the modified Rodrigues parameters which move the singularity to rotations equal to 360.

\[
\mathbf{p} = \frac{q}{1 + q_4} \tag{2.31}
\]

The Rodrigues parameters can be seen as a set of stereographic (projection of a sphere into a plane) orientation parameters. Stereographic projections are used to map the higher-dimensional spherical four dimensional surface, where the quaternions are defined, into a lower dimensional tree dimensional hyperplane. The point of projection is \( q = [0, 0, 0, 0] \) and the parameters are projected into the tangent hyperplane at \( q_4 = 1 \). For the case of the modified Rodrigues parameters, the projection point is \( q = [0, 0, 0, -1] \) with the projection at plane \( q_4 = 0 \). The quaternion and Rodrigues parameters have a set of shadow parameters which are equivalent in representing a rotation but differ in a sign in their representation \( q = -q \), this is not the same for the modified Rodrigues parameters as can be seen in figure [2].

2.5.1 Modified Rodrigues composition

The composition rule for modified Rodrigues parameters can be derived substituting (2.31) into the quaternion composition (2.15)

\[
\mathbf{p}'' = \mathbf{p}' \otimes \mathbf{p} = \frac{(1 - \mathbf{p}^\top \mathbf{p})\mathbf{p}' + (1 - \mathbf{p}'^\top \mathbf{p}')\mathbf{p} - 2\mathbf{p}' \times \mathbf{p}}{1 + (\mathbf{p}^\top \mathbf{p})(\mathbf{p}'^\top \mathbf{p}') - 2\mathbf{pp}'^\top} \tag{2.32}
\]

2.5.2 Modified Rodrigues evolution with time

The time derivative of the modified Rodrigues parameters is derived directly by substituting (2.31) into (2.20)
\[
{L(t) \over G} \mathbf{p}(t) = \frac{1}{2} \left[ S(p) + pp^\top + \left( \frac{1 - p^\top p}{2} \right) \right] w
\]
\[
= \frac{1}{2} B(p) w
\]

where \( S(p) \) is defined as

\[
S(p) = \begin{pmatrix}
0 & -p_3 & p_2 \\
p_3 & 0 & -p_1 \\
-p_2 & p_1 & 0
\end{pmatrix}
\]

(a) Stereographic projection for rodrigues parameters
(b) Stereographic projection for modified rodrigues parameters

Figure 2.2: Graphical representation of the relation between quaternions, rodrigues and modified rodrigues parametrization
3 Optimal state estimation

In this chapter we will review the basics of optimal filtering and estimation with a tool called the Kalman filter. The quantities that we are going to work with are random variables and they have a measure of confidence associated with there value. There does not exist perfect sensors in the world making it impossible to have a hundred percent reliable measure, that is why with each sensor reading we associate a value and uncertainty. Using different sensors with different error characteristics we can try to merge them taking into account their strengths and weaknesses. A complete example of these will be shown in the derivation of the inertial measuring unit that tracks the human’s head in order to move the robot’s head accordingly. Also the term filter usually comes up with the Kalman filter when it is also an estimator. An example of this is the designed human skeleton filter that not only is used to smooth the original signal from the camera feature tracker but also to estimate the joint angles of the human limbs (since the tracker only gives the Cartesian position of each limp).

We will focus on a particular distribution to describe the density function of a random variable, which is the Gaussian distribution. It can be parametrized with two variables, the mean \( \mu \) and the variance \( \sigma^2 \). The function that describes it for the scalar case is

\[
f(x; \mu, \sigma^2) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{1}{2}(x-\mu)^2}
\]

and its multivariable form with the vector mean \( \mu \) and its associated covariance matrix \( \Sigma \)

\[
f_{\mathbf{x}}(x_1, \ldots, x_k) = \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp \left( -\frac{1}{2} (\mathbf{x} - \mu)^\top \Sigma^{-1} (\mathbf{x} - \mu) \right)
\]

3.1 Transformation of vector random variables

Random variables can be transformed by functions. An example would be the estimation of the location of a robot and how it changes as a function of time and control input. If the function is linear "\( \mathbf{y} = A\mathbf{x} \)" it is very easy to apply. The resulting random variable can be obtained as

\[
f_{\mathbf{y}}(y_1, \ldots, y_k) = \frac{1}{(2\pi)^{N/2}|A\Sigma A^\top|^{1/2}} \exp \left( -\frac{1}{2} (\mathbf{y} - A\mu)^\top (A\Sigma A^\top)^{-1} (\mathbf{y} - A\mu) \right)
\]

which is also a Gaussian variable \( N(A\mu, A\Sigma A^\top) \). In the case that the function is not linear we will do a linear approximation of the function and use it to transform
the variable. Usually the estimate of the mean of the distribution is used as the point of linearization.

### 3.2 Maximum likelihood estimation

Given a set of noisy estimates of the real value that we are searching for, we can find its value by maximizing its likelihood with respect to the information that we have. Each measurement has an uncertainty associated in the form of additive noise and by combining them we will end with an estimate of the real value that we are searching and how certain of it we are.

From now on we will write (3.3) in a more simplified way as

\[
P(x) \propto \exp \left( -\frac{1}{2} \|Hx - b\|_\Sigma^2 \right) = \prod f \exp \left( -\frac{1}{2} \|H_i x - b_i\|_\Sigma^2 \right) \tag{3.4}
\]

where \( \propto \) means proportional since we are evading the normalizing factor and \( \|.\|_\Sigma^2 \) is the squared Mahalanobis distance. As the number \( n \) of variables grows inside a distribution it becomes more computationally expensive to work with it, but thanks that inside the distribution we have independence assumptions between the variables (in a Bayesian network given the values of the parents of a conditional variable we can infer the distribution for that variable independently of the rest of the network), we can rewrite the distribution as a product of independent distributions

Because we want to maximize the probability we will maximize the negative logarithm of the equation with respect to the variable to get a linear expression. Doing this we get rid of the \( \exp \) and transform the product of the individual distributions into a sum

\[
\|Hx - b\|_\Sigma^2 = \sum_i \|H_i x - b_i\|_\Sigma^2 \tag{3.5}
\]

Minimizing (3.5) is equivalent to minimize the cost function (3.6). The matrix \( W \) is equal to \( \Sigma^{-1} \).

\[
J(\hat{x}) = \frac{1}{2} (\hat{Y} - H\hat{x})^\top W (\hat{Y} - H\hat{x}) \tag{3.6}
\]

Just as they are defined in (3.4) the matrix \( H \) and vector \( Y \) is composed of the individual factors of the distribution.

\[
\hat{Y} = [\hat{y}_1, \ldots, \hat{y}_m] \text{ and } H = [H_1^\top, \ldots, H_m^\top].
\]
To minimize the cost function (3.6) we take its derivative and make it zero. Solving the system we can find the optimal value of \( x \), which turns out to be the weighted least square solution.

\[
\frac{\partial J}{\partial \hat{x}} = -H^T W \hat{Y} + H^T W x = 0
\] (3.7)

\[
E[\hat{x}] = \hat{x} = (H^T WH)^{-1} H^T W \hat{Y}
\] (3.8)

To find out the covariance matrix \( P = E[(x - E[x]) (x - E[x])^\top] \) associated with the estimate of \( \hat{x} \) found in (3.8) we will first define the following error vector between the real value of \( x \) and its estimate \( \hat{x} \) using the fact that \( \hat{Y} = Hx + \sigma \), where \( E[\sigma] = 0 \) and \( E[\sigma \sigma^\top] = \Sigma \)

\[
\delta x = x - (H^T WH)^{-1} H^T W \hat{Y}
\]
\[
= x - (H^T WH)^{-1} H^T W (Hx + \sigma)
\]
\[
= (I - (H^T WH)^{-1} H^T WH)x - (H^T WH)^{-1} H^T W \sigma
\]
\[
= -(H^T WH)^{-1} H^T W \sigma.
\] (3.9)

We note that \( \delta x \) is independent of the measurement vector \( \hat{Y} \) and the value of \( x \). Because we have the assumption that the noise added to the system is Gaussian with zero mean we can assume the following properties

\[
E[\delta x] = 0
\] (3.10)

\[
var[\delta x \cdot \delta x^\top] = P = (H^T WH)^{-1} H^T W \Sigma WH (H^T WH)^{-1}
\]
\[
= (H^T \Sigma H)^{-1}
\] (3.11)

given the fact that

\[
P = var[\hat{x} \cdot \hat{x}^\top] = var[\delta x \cdot \delta x^\top]
\] (3.12)

It can be noted that (3.11) is how the covariance transforms under the inverse function.

### 3.3 SAM (Simultaneous smothing and mapping) example

To clarify this we give a simple example with a problem called simultaneous localization and mapping [2]. We have a robot in 2D that moves with a very simple
odometry, the state of the robot has the form \( x = [x \ y] \) which describes a 2D position. The robot has a sensor that allows it to measure the euclidean distance to an obstacle. Merging the information of the distance sensor and the odometry we can reconstruct the original path followed by the robot (which originally was corrupted by noise in the encoders of the robot). We have a distribution \( x_i = f(x_{i-1}, u) \) that describes the new position of the robot given a past state and its odometry. We also have a distribution \( z_k = h(x_{i_k}, l_j) \) over the distance from a landmark to a given position of the robot. Joining all this probabilities we get a Bayesian net that represents our problem. Using (3.4) we can write the problem in matrix form and find the optimal solution by using least squares. Due to the independencies in the graph the resulting matrix is sparse, making the problem much more tractable compared to the case where it would be dense. The distribution that describes the problem is (3.13) where we have the robots successive positions, the landmarks and the measurements as random variables. The distribution can be factored into a prior for the initial guess of the robots position in the world, the successive odometry factors that are conditionally dependent on the previous position and the measurement factors that are conditionally dependent on the position of the robot and the landmark that corresponds the measurement.

\[
P(X, L, Z) = P(x_0) \prod_{i=1}^{N} P(x_i|x_{i-1}, u_i) \prod_{k=1}^{K} P(z_k|x_{i_k}, l_{jk}) \tag{3.13}
\]

each factor describes the corresponding gaussian process, where \( \sigma_i \) and \( v_i \) is additive Gaussian zero mean noise

\[
x_i = f(x_{i-1}, u_i) + \sigma_i \iff P(x_i|x_{i-1}, u_i) \propto \exp -\frac{1}{2} \| f_i(x_{i-1}, u_i) - x_i \|^2 \tag{3.14}
\]

\[
z_k = h(x_{i_k}, l_{jk}) + v_i \iff P(z_k|x_{i_k}, l_{jk}) \propto \exp -\frac{1}{2} \| h_k(x_{i_k}, l_{jk}) - z_k \|^2 \tag{3.15}
\]

with the function defined as

\[
\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix} \tag{3.16}
\]

and measurement model

\[
z = \sqrt{(x_x - l_x)^2 + (x_y - l_y)^2} \tag{3.17}
\]

Combining all the linearised odometry and measurement factors gives a matrix that looks like (3.18). Each individual factor has been scaled according to the Mahalanobis distance defined by its covariance matrix \( \| e \|^2_\Sigma = e^T \Sigma^{-1} e \). A real example
with simulated data is shown in figure 3.1 where a robot with noisy odometry follows a trajectory and combing information from the sensors is able to recover the real path that is has travelled.

\[
A = \begin{bmatrix}
G_1^1 & F_2^1 & G_2^2 \\
F_3^1 & G_3^3 \\
H_1^1 & J_1^1 \\
H_2^1 & J_2^2
\end{bmatrix}, \quad b = \begin{bmatrix}
a_1 \\
a_2 \\
a_3 \\
c_1 \\
c_2
\end{bmatrix}
\] (3.18)

In the above matrix \(G\) represent \(-I_{n \times n}\) with \(n\) the size of the state. \(F\) and \(G\) are the Jacobians of the odometry and measurement factors as in that arise when linearising the non-linear functions of each factor. The vector \(b\) corresponds to the measurement and prediction error \(a_i = x_i - f(x_{i-1}, u_i)\). Iteratively solving the system and updating the state with the estimate error ends up giving the optimal result. A simulated example with the corrupted and corrected estimate is shown in figure 3.1.
Figure 3.1: SAM example where yellow indicates the robot’s real path, blue is the robot’s odometry, and red are the measurements to the landmarks.
### 3.4 Recursive least squares

Since we are continuously receiving new information we don’t want to solve the problem all over again at every step. There should be a way to use the previous solution found and update it with the new information we have just gotten. The solution to this is the recursive least squares in which we have a current estimate of the solution $\hat{x}_m$ up until time $m$ and we update it with the new information $\hat{y}_{m+1}$ at $m+1$. We are seeking for a solution of the form

$$\hat{x}_{m+1} = \hat{x}_m + K_{m+1}(\hat{y}_{m+1} - \hat{y}_{m+1})$$  \hspace{1cm} (3.19)

the new estimate $\hat{x}_{m+1}$ at time $m+1$ is the solution to

$$\begin{bmatrix} \hat{x}_m \\ \hat{y}_{m+1} \end{bmatrix} = \begin{bmatrix} I \\ H_{m+1} \end{bmatrix} x + \begin{bmatrix} -\delta x_m \\ \sigma_{m+1} \end{bmatrix}$$  \hspace{1cm} (3.20)

The new associated covariance matrix is found using (3.11) which gives

$$P^{-1}_{m+1} = \begin{bmatrix} I & H_{m-1}^T \\ H_m & \Sigma_{m+1} \\ \Sigma_{m+1} & H_m \end{bmatrix} P_m \begin{bmatrix} 0 \\ \Sigma_{m+1} \\ H_m \end{bmatrix}$$ \hspace{1cm} (3.21)

To solve the system we first will introduce some manipulations. We multiply (3.21) on the right by $\hat{x}_m$, and we obtain

$$P_{m+1}^{-1} \hat{x}_m = P_{m+1}^{-1} \hat{x}_m + H_{m+1}^{T} \Sigma_{m+1} \hat{y}_{m+1}$$ \hspace{1cm} (3.22)

Now multiplying on the left by $P_{m+1}$ we get

$$P_{m+1} P^{-1}_{m} \hat{x}_m = \hat{x}_m - P_{m+1} H_{m+1}^{T} \Sigma_{m+1} \hat{y}_{m+1}$$ \hspace{1cm} (3.23)

which will be needed in the next derivation. We are ready to solve the system (3.19) which will give the next best estimate, and the way we will solve the system is by using (3.18)

$$x_{m+1} = P_{m+1} \left[ I \ H_{m+1}^{T} \right] \begin{bmatrix} P_m \\ \Sigma_{m+1} \end{bmatrix} \begin{bmatrix} \hat{x}_m \\ \hat{y}_{m+1} \end{bmatrix}$$

$$= P_{m+1} (P^{-1}_m \hat{x}_m + H_{m+1}^{T} \Sigma_{m+1} \hat{y}_{m+1})$$

$$= x_m + P_{m+1} H_{m+1}^{T} \Sigma_{m+1} (\hat{y}_{m+1} - \hat{y}_m)$$

$$x_{m+1} = \hat{x}_m + K_{m+1}(\hat{y}_{m+1} - \hat{y}_m)$$
where we have defined $K_{m+1} = P_{m+1}H_{m+1}^T\Sigma_{m+1}^{-1}$. Surprisingly we have arrived to a form equal to the one we where seeking (3.14), a simple closed form to recursively add the new information to the last estimate. From the second to the last line of the derivation we have used (3.23).

Because (3.21) requires an inversion to get the result we will modify the equations using the matrix inversion lemma to get rid of the inversion of $P_{m+1}$ and the resulting final equations become

$$K_{m+1} = P_{m}H_{m+1}^T(\Sigma_{m+1} + H_{m+1}PH_{m+1}^T)^{-1}$$ (3.25)

$$\hat{x}_{m+1} = x_m + K_{m+1}(\tilde{y}_{m+1} - H_{m+1}x_m)$$ (3.26)

$$P_{m+1} = (I - K_{m+1}H_{m+1})P_m$$ (3.27)

### 3.5 Kalman filter

We are now ready to derive the Kalman filter as an extension to recursive least squares. The Kalman filter solves the problem of estimating the state of a dynamic system described by an ordinary differential equation. In other words, we are trying to solve the bayes net of figure 3.2. The Kalman filter makes the assumption that all the information needed to estimate the current state is the previous state of the system and the current information (measurements) of it. It’s similar to solving the SAM problem but we are only interested in the current state of the robot, we do not care about its past states or anything that is not related to the state of the robot. The way this is done is by first updating the state according to the differential equation that governs the system getting a new state $\hat{x}_{t+1}$ and covariance matrix $\Sigma_t$:

$$\hat{x}_{t+1} = \Phi\hat{x}_t + w_t$$ (3.28)

$$\Sigma_{t+1} = \Phi\Sigma_t\Phi^T + Q_t$$ (3.29)

and correcting it comparing the prediction of what we should measure in the current state $\hat{x}_{t+1}$ and what we really are measuring $\tilde{y}$, to get final state $x_{t+1}^\ast$. This last part can be describe as a recursive least square problem

$$\hat{x}_{t+1} = x_{t+1}^\ast - \delta x_{t+1}$$

$$\tilde{y}_{t+1} = Hx_{t+1}^\ast + v_{t+1}$$ (3.30)
Figure 3.2: Simple bayes net that represents a Kalman filter problem

where $\mathbf{x}_{t+1}$ is the state before incorporating the new measurement with $\delta \mathbf{x}_{t+1}$ the prediction error with $\text{var}[\delta \mathbf{x}_{t+1}] = \Sigma_{t+1}$. We will make the following assumptions about the noise

$$E[ww^\top] = Q\delta$$  \hspace{1cm} (3.31)

$$E[vv^\top] = R\delta$$  \hspace{1cm} (3.32)

$Q_t^d$ is the integrated noise over the same time step that we use to integrate our system

$$Q_t^d = \int_t^{t+1} \Phi(t + 1, \tau)Q\Phi(t + 1, \tau)^\top d\tau$$  \hspace{1cm} (3.33)

which for small time intervals we can make the following approximation whenever the eigenvalues of $\Phi$ are very small compared to the period of integration

$$Q_t^d \approx Q_t^c \Delta t$$  \hspace{1cm} (3.34)

Putting it all together we describe the Kalman filter equations in table 3.1. A graphic representation is shown in 3.2.
Kalman Filter

1. \( x_{t+1}^- = \Phi_t x_t + w_t \)
2. \( \Sigma_{t+1}^- = \Phi \Sigma_t \Phi^T + R_t \)
3. \( K_{t+1} = \Sigma_{t+1}^- H_{t+1}^T (Q_{t+1} + H_{t+1} \Sigma_{t+1}^- H_{t+1}^T)^{-1} \)
4. \( x_{t+1}^+ = x_{t+1}^- + K_{t+1} (y - H x_{t+1}^-) \)
5. \( \Sigma_{t+1}^+ = (I - K_{t+1} H_{t+1}) \Sigma_{t+1}^- \)

Table 3.1: Linear Kalman Filter

3.5.1 Kalman filter example

Given a noisy position measurement we are going to try to filter out the noise and at the same time we are going to estimate its velocity. We are going to compare the results by doing the same thing using a low pass filter and finite differences to find the velocity of the signal. The system matrix is (3.35) and measurement matrix

\[
\begin{bmatrix}
  x_{t+1} \\
  v_{t+1} \\
  a_{t+1}
\end{bmatrix} = \begin{bmatrix}
  1 & \Delta t & \frac{1}{2} \Delta t^2 \\
  0 & 1 & \Delta t \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x_t \\
  v_t \\
  a_t
\end{bmatrix} + n \tag{3.35}
\]

We have the following assumptions about \( n \)

\[
E[n] = 0_{3x1}
\]

\[
E[n \cdot n^T] = \begin{bmatrix}
  0 & 0 & 0 \\
  0 & 0 & 0 \\
  0 & 0 & 1
\end{bmatrix} \sigma \tag{3.36}
\]

which in essence means that the only uncertainty in our system is in the acceleration. It can be viewed that our system is driven by noise at the acceleration level. The measurement matrix is

\[
y = \begin{bmatrix}
  1 & 0 & 0
\end{bmatrix} + \sigma \tag{3.37}
\]

\[
E[\sigma_r] = 0
\]

\[
E[\sigma_r \sigma_r^T] = \sigma^2 \tag{3.38}
\]
The results are illustrated in figure 3.3. The top row shows the original signal with its KF filtered and low pass filtered version. To the right it can be seen that the KF has half of the error. The bottom row shows the original velocity of the signal compared with the two output of the two filters. The KF treats the velocity as an unknown state while it tries to estimate it and the low pass filter guesses the velocity using finite differences. It can be seen that the velocity estimate from the low pass filter is practically unusable manifesting the dramatic difference of using optimal filtering techniques (it also has to be noted that the low pass filter has completely cut off all the high frequency components that existed in the original signal).
4 Orientation estimation and filtering

The body part tracker that we are using is the skeleton filter from the Openni Middleware that is integrated into ROS. This tracker outputs a reference frame with respect to the camera sensor for each of the fourteen body parts that it tracks. The tracker relies in a noisy depth image which makes the tracking noisy and giterish. This added noise (which we will assume that is Gaussian) manifests as vibrations in the robot’s teleoperation pipeline. The solution proposed is to use a Kalman filter for the linear translation part \( [x, y, z] \) and an Extended Kalman filter for the orientation \( SO(3) \). We are going to make the assumption that the frame moves with constant acceleration and constant angular velocity. The tracker outputs the orientation of the frame parametrized with a quaternion but as we said in §2.4, they cannot be used for estimation. That’s why we will switch to the modified Rodrigues parametrisation and switch back to quaternions once the filtering is done. In this chapter we will explain how to use and derive a rotation filter and an extension to the Kalman filter called the Extended Kalman filter that is used when either or both of the driving or measuring functions that describe our function is non linear.

4.1 Extended Kalman Filter

The EKF is used when the functions that describe the state evolution and the measurements are not linear. The way the EKF treats this non linearity is to linearise this functions around the current estimate of the mean and covariance of the variables using a Taylor expansion. Once this is done the usual Kalman equations can be used with the linearised functions.

Linearization

Given a nonlinear function \( \mathbf{x} = f(x, y) \) we can obtain a linear approximation of it using a Taylor expansion centered at \( x = \hat{x}_0, y = \hat{y}_0 \)

\[
\begin{align*}
\mathbf{x} = f(x, y) & \approx f(\hat{x}_0, \hat{y}_0) + \frac{\partial f(x, y)}{\partial x} \bigg|_{x=\hat{x}_0, y=\hat{y}_0} (x - \hat{x}_0) + \frac{\partial f(x, y)}{\partial y} \bigg|_{x=\hat{x}_0, y=\hat{y}_0} (y - \hat{y}_0) \\
& = f(\hat{x}_0, \hat{y}_0) + \mathbf{F} \left( \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} \hat{x}_0 \\ \hat{y}_0 \end{bmatrix} \right)
\end{align*}
\]

(4.1)

and the first order equation that describes the transformation of the error can be approximated as

\[
\delta \dot{\mathbf{x}} = f(x, y) - f(\hat{x}_0, \hat{y}_0) = \mathbf{F} \left( \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} \hat{x}_0 \\ \hat{y}_0 \end{bmatrix} \right) = \mathbf{F} \delta \mathbf{x}
\]

(4.2)
Extended Kalman Filter

1. $x_{t+1} = f(x_t) + w_t$
2. $\Sigma_{t+1}^- = FF_t^T + R_{t+1}$
3. $\hat{y} = h(x_{t+1})$
4. $K_{t+1} = \Sigma_{t+1}^- H_t^T (Q_{t+1} + H_t \Sigma_{t+1}^- H_t^T)^{-1}$
5. $x_{t+1}^+ = x_{t+1}^- + K_{t+1}(\hat{y} - \hat{y})$
6. $\Sigma_{t+1}^+ = (I - K_{t+1} H_t) \Sigma_{t+1}^-$

Table 4.1: Extended Kalman Filter

Given a linearization of the system and measurement equations the extended Kalman filter is summarized in Table 4.1. The only differences with the original KF are in 1. where we use the non-linear function to evolve the system and its Jacobian to update the covariance in 2. The same thing applies to the measurement in 3. its update of the covariance in 4. and 5. where the Jacobian of the measurement is used.

4.2 Filter implementation and results

The equations that drive our system are (4.3) where $mdr(p, w)$ is (2.33) and $\sigma$ is the white Gaussian noise that drives our system. The state of the system will be defined as $x = [p, w]$. Since the system is non linear it cannot be expressed in matrix form. A solution to it cannot be found using matrix exponentiation. We will use a simple euler integration to discretize the system knowing that error can be introduced due to imprecise integration. The resulting system is

\[
\dot{x} = f(p, w) = \begin{bmatrix} \dot{p} = mdr(p, w) \\ \dot{w} = 0_{3x1} \end{bmatrix} + w_i \tag{4.3}
\]

\[
y = \begin{bmatrix} I_{1x3} & 0_{1x3} \end{bmatrix} x + v_i \tag{4.4}
\]

where we make the following assumption

\[
E[w_r] = 0_{3x1}
\]

\[
E[w_r w_r^T] = \begin{bmatrix} 0_{3x3} & 0_{3x3} \\ 0_{3x3} & I_{3x3} \end{bmatrix} \sigma \tag{4.5}
\]
The Jacobian $\frac{\partial f(x)}{\partial x}$ of (4.3) needed for the EKF is

$$
F = \frac{\partial f(x)}{\partial x} = \begin{bmatrix}
\frac{1}{2}[pw^\top - wp^\top - [w\times] + w^\top wI_{3x3}] & \frac{1}{2}B(p) \\
0_{3x3}
\end{bmatrix}
$$

(4.6)

Using an euler integration gives the discretized system (4.7). A more precise integration could be done like for example using a Runge-Kutta method.

$$
p_{t+1} = p_{t} + mdr(p_{t}, w_{t}) \Delta t + v \Delta t
$$

$$
w_{t+1} = w_{t} + v \Delta t
$$

(4.7)

a similar integration works for discretizing the Jacobian. Once we have the filtered state we can recover the original quaternion parametrization using

$$
q_i = \frac{2p_i}{1 + p^2} \quad i = 1 \ldots 3
$$

$$
q_0 = \frac{1 - p^2}{1 + p^2}
$$

(4.8)

4.2.1 Simulation

We create an angular velocity signal parametrized by

$$
x = 0.1 \cos(t)(2 - \cos(\frac{2t}{3}))
y = 0.1 \sin(t)(2 - \cos(\frac{2t}{3}))
z = 0.1 - \sin(\frac{2t}{3})
$$

(4.9)

We will integrate this angular velocity with an added Gaussian zero mean noise with $\Sigma = 1.5$ and filter with the EKF to obtain the results of figure 4.1 for the orientation and 4.2 for the estimated angular velocity that is a hidden variable in the system. At each step noise is added in the angular velocity, but the starting point for integration is set to original signal. This allows us to keep track of the original signal for later comparison.
Figure 4.1: Results of the Rodrigues EKF estimating and filtering a quaternion

5 Human skeleton estimation and robot teleoperation

There are many types of body trackers. They can be broadly classified into the ones that use classification or tracking techniques. The first ones try to individually classify the pixels of the image to corresponding labels [8], while the second ones try to track and fit a skeleton to the image [5]. The classifying approach can robustly detect a person independently of the previous frames in time. They do this by classifying all the pixels of the image at each frame in time. Because they don’t use any type of temporal information they may miss classify background pixels in the image for human body parts. The tracking approach in the other hand discriminates in the image the person using temporal information, but has problems in recovering if they loose the person that where tracking, making them less robust.

In the filter that we are going to develop in this chapter we are interested in obtaining a smooth signal that tracks and estimates the human skeleton position and its joint angles. We will use the position of the body limbs with respect to the sensor frame that the Openni sdfasfddsf tracker outputs as measurements. The goal is to smooth the position measurements and at the same time obtain the joint
Figure 4.2: Results of the Rodrigues EKF estimating the angular velocity of human skeleton. The way we will do this is by assuming that the human joint angles move at constant velocity. The system that describes the way the joint angles evolve is similar to the system that describes how the translational part evolves in the filter created in the previous chapter, its a system of integrators. To correct the joint angles we will develop a mapping between the humans joint angles and their position in Cartesian space using forward kinematics. This filter belongs to the tracking class. We will first explain how we design our measuring function given the skeleton’s state in order to iteratively correct our estimate, later we will also explain another extension of the Kalman filter that is more suited when we have non linear functions called the unscented Kalman filter. Finally we will explain how the tracked skeleton can be used to teleoperate a robot.

5.1 Denavit-Hartenberg

The Denavit-Hartenberg parameters offer a systematic way to describe and transform between the different frames that correspond to the links of a robot and also imposes a convention among them. Each link has either a translational or revolute joint that points in the z direction, the x direction...
is defined by the intersection of the z axis of two consecutive frames and finally the y direction comes out of the cross-product of the last two. The reason this parametrization was chosen is the easiness of describing kinematic chains, and the minimum number of parameters that needs to be defined. Having a minimum set of parameters that describe the state lets us minimize the computational burden and increase the estimation speed.

Each link is described by four parameters \([a \ d \ \alpha \ \theta]\) which describe an homogeneous transformation between the actual link and the next one. The meaning of each parameter can be seen in figure 5.1.

\[
T_n = \begin{bmatrix}
\cos \theta_n & -\sin \theta_n & 0 & a_n \\
\sin \theta_n \cos \alpha_n & \cos \theta_n \cos \alpha_n - \sin \alpha_n & \cos \alpha_n & -d \sin \alpha_n \\
\sin \theta_n \sin \alpha_n & \cos \theta_n \sin \alpha_n & \cos \alpha_n & d_n \cos \alpha_n \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

(5.1)

Table 5.3 shows the description of a human skeleton using DH. The state has 40 dimension (20 for the joint angles and 20 for the velocity of the joints) and makes it computationally difficult to filter, that is why we will use a simplified
model that only uses the upper two tables that correspond to each arm of the skeleton (the pelvis and vertebra will also be ignored), a graphical representation of full skeleton and its top torso simplification can be seen in figure 5.2

5.2 Uncented Kalman filter

They way the EKF treats the non linearity is often not right since the new liberalization might be very imprecise in relation of how the real function behaves. Also calculation the Jacobian is often complicated and costly and concretely in our case can become singular. The UKF is an alternative filter that parametrizes the Gaussian distribution into sigma points (A set of $2L + 1$ points that describe the original Gaussian where $L$ is the dimension of the state vector). Each of the
Figure 5.3: DH-table representing a human skeleton. Distances between link representations body proportions, they are labeled with the name of their associated links.
sigma points is represented by a tuple \( \mathcal{X}_i, W_i \) and they are derived from a current estimate \( \hat{x} \) as

\[
\begin{align*}
\mathcal{X}_0 &= \hat{x} \\
\mathcal{X}_i &= \hat{x} + \left( \sqrt{(L + \lambda) \Sigma} \right)_i \\
\mathcal{X}_i &= \hat{x} - \left( \sqrt{(L + \lambda) \Sigma} \right)_{i-L}
\end{align*}
\]

\[
W^{(m)}_0 = \lambda(L + \lambda) + (1 - \alpha^2 + \beta)
\]

\[
W^{(c)}_0 = \lambda/(L + \lambda) + (1 - \alpha^2 + \beta)
\]

\[
W^{(m)}_i = W^{(c)}_i = 1/2(L + \lambda) \tag{5.3}
\]

where \( \lambda = \alpha^2(L + \kappa) - L \) is a scaling parameter, \( \alpha \) determines the spread of the sigma points around \( \hat{x} \) (usually set to 1e-3), \( k \) is a second scaling (usually set to 0), and \( \beta \) is for incorporating prior information about the prior distribution (for Gaussian distributions \( \beta = 2 \) is optimum). The mean and covariance of the original distribution can be recovered as

\[
\begin{align*}
\hat{x} &\approx \sum_{i=0}^{2L} W^{(m)}_i \mathcal{X}_i \\
\Sigma &\approx \sum_{i=0}^{2L} W^{(c)}_i \{ \mathcal{X}_i - \hat{x} \} \{ \mathcal{X}_i - \hat{x} \}^T
\end{align*}
\]

They way the distribution transforms under the new parametrization is by applying the nonlinear function (5.5) to each of the vector sigma points. From the new sigma points we can recover the transformed distribution using (5.4)

\[
\mathcal{Y}_i = g(\mathcal{X}_i) \quad i = 0, \ldots, 2L \tag{5.5}
\]

The UKF has similarities to particle filters in the way the distributions are treated. The difference is that the UKF uses a fixed number of points contrary to the particle filter where the number of particles is chosen depending on the precision that wants to be achieved. The UKF can be thought as a smart way of choosing \( 2L + 1 \) particles in problems where we can make the assumption that the underlying distribution is Gaussian and the system and measurement equations are not linear. Particle filters in the other hand can also work with variables that have arbitrary density models but at a much computational expense.

Now we can transform the original KF into the UKF by changing the parametrization used. The result is showed in table 5.1.
Uncented Kalman Filter

1. $\mathbf{x}_t = [\bar{x}_t \bar{x}_t \pm (\sqrt{(L + \lambda)\Sigma_t})]$

2. $\mathbf{x}_{t+1}^- = f(\mathbf{x}_t) $

3. $\mathbf{x}_{t+1}^- = \sum_{i=0}^{2L} W_i^{(m)} \mathcal{X}_{t+1|i}$

4. $\mathbf{\Sigma}_{t+1}^- = \sum_{i=0}^{2L} W_i^{(c)} \{ \mathcal{X}_{t+1|i} - \bar{x} \} \{ \mathcal{X}_{t+1|i} - \bar{x} \}^\top + \mathbf{R}_{t+1}$

5. $\mathbf{z}_{t+1} = h(\mathbf{x}_{t+1}^-)$

6. $\mathbf{z}_{t+1} = \sum_{i=0}^{2L} W_i^{(m)} \mathcal{Z}_{t+1|i}$

7. $\mathbf{S}_{t+1} = \sum_{i=0}^{2L} W_i^{(c)} \{ \mathcal{Z}_{t+1|i} - \bar{z} \} \{ \mathcal{Z}_{t+1|i} - \bar{z} \}^\top + \mathbf{Q}_{t+1}$

8. $\mathbf{\Sigma}_{t+1}^{x,z} = \sum_{i=0}^{2L} W_i^{(c)} \{ \mathcal{X}_{t+1|i} - \bar{x} \} \{ \mathcal{Z}_{t+1|i} - \bar{z} \}^\top$

9. $\mathbf{K}_{t+1} = \mathbf{\Sigma}_{t+1}^{x,z} \mathbf{\Sigma}_{t+1}^{-1}$

10. $\mathbf{x}_{t+1}^+ = \mathbf{x}_{t+1}^- + \mathbf{K}_{t+1}(\bar{y} - \mathbf{z}_{t+1})$

11. $\mathbf{\Sigma}_{t+1}^+ = \mathbf{\Sigma}_{t+1}^- + \mathbf{K}_{t+1} \mathbf{S}_{t+1} \mathbf{K}_{t+1}^\top$

Table 5.1: Uncented Kalman Filter
5.3 Filter implementation and results

In order to teleoperate the robot we only need the state of the upper torso of the human. To reduce computational burden we will develop a skeleton tracker for the upper torso. Two identical filters will be developed, one for each arm of the user. The root of each kinematic chain will be the human pelvis and the tip will be each of two hands. Since we are using the UKF we only need to worry about defining our system propagation and measurement function.

The state evolves according to chain of integrators

\[
\begin{bmatrix}
\theta_1 \\
\vdots \\
\theta_n \\
\end{bmatrix}
= 
\begin{bmatrix}
1 & 0 & 0 & \cdots & \Delta t & 0 & \cdots & 0 \\
0 & 1 & 0 & \cdots & 0 & \Delta t & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & 0 & 0 & \cdots & 1 & 0 & \cdots & \Delta t \\
0 & 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \\
\end{bmatrix}
\begin{bmatrix}
\theta_1 \\
\vdots \\
\theta_n \\
\end{bmatrix}
+ \mathbf{v}
\] (5.6)

where \( \mathbf{v} \) is assumed to be zero mean gaussian noise. For each arm we have a four dimensional integrator (tree degrees for the shoulder and one degree for the elbow)

\[
x_{t+1[leftarm]} = \text{integrator}(x_{t[leftarm]})
\]

\[
x_{t+1[rightarm]} = \text{integrator}(x_{t[rightarm]})
\] (5.7)

The measurement function is the forward kinematics describe in the DH table of each arm given the joint state angles

\[
y_{[leftarm]} = f_{[leftarm]}(x_{t+1})
\]

\[
y_{[rightarm]} = f_{[rightarm]}(x_{t+1})
\] (5.8)

The results of the filter are shown in figure. The top figure shows the result of filtering the position of the right hand, the bottom figure show the estimated angles of the right arm. It has to be noted that estimating the shoulder state is the same as estimating a tree dimensional rotation, since is has 3 degrees of freedom that define a reference frame.

5.4 Teleoperation

There are many ways to describe the human skeleton using the DH parameters, out of all of them the one that maps the humans joint angles directly to the robot
was chosen (up to sign ambiguity, having the same axis of rotation for the joints in the two systems). This allows us to imitate the hands of the user in the robot and because we have a redundant manipulator map in the null space of the robot the joint angles of the human. This way we are able to track the users hands trajectory with the robot and get at the same time the same expressiveness in the robot as the human. Because we have a parametrized skeleton fitted to the user we can re-target this skeleton by rooting it in the robots reference frame and changing its proportions in order to match the robots.

Given the joint angles of the robot we can get the position of its link using forward kinematics.

\[ x = FK(\theta) \]  

(5.9)

Doing the inverse mapping is more difficult because we have a redundant manipulator and as result a Cartesian position can mapped to multiple joint configurations. Because the function is non linear we cannot invert it, we will work with its first derivative that gives a linear mapping between joint velocities and end effector Cartesian velocities. Up until now we were facing the opposite problem, where there wasn’t a solution to the system and we solved it finding a solution that minimized the squared root error. Now we have multiple solution to the system, all of them valid, a first approach is to choose the one with smaller norm among them. A more sophisticated way is to choose inside the set of possible solutions the one that represents better the state of the users skeleton in the robot. We solve this problem using inverse kinematics with null space optimization, the first task is to to track the users hands with the robot’s, and as a secondary task the joint angles of the robot must resemble as much as they can the users.

First state first order kinematics

\[ \dot{x} = \left( \frac{FK(\theta)}{\delta \theta_i} \right) = J \dot{\theta} \]  

(5.10)

Because we have an undetermined system \( J \) is\((m < n)\) we find the least norm solution solution using the right pseudo inverse.

\[ \dot{\theta} = J^\dagger \dot{x} = J^T(JJ^T)^{-1} \dot{x} \]  

(5.11)

Using a discretized version of the above equation we will use a term proportional to the error in desired position as the desired velocity in the system. Depending on the configuration of the arm we can arrive at singular configurations. To avoid numerical instability we will add a regularizing term to the equation and sum a second objective to the function that resides in the null space of the primary task (Inside the set of possible solutions we optimize for a particular one)
\[
\Delta \theta = J^T(JJ^T + \lambda^2 I)^{-1}\Delta x + J(I - J^TJ)\phi
\] (5.12)

Using the outputs of the filter in the above equation gives a periodical update to the angles of the robot letting us teleoperate the robot using the humans movements. \(\Delta x\) is the tracking position error, \(\lambda\) is the regularization parameter that tries to keep the norm of the solution small and \(\phi\) is the error in the null space objective (the desired joint angles).

### 6 Attitude heading and reference system & robot head and wrist control

We can use sensors to estimate the attitude of an object in space. The object can be an aeroplane, a robot or, in our case, a human limp. None of the open available image body trackers gives the orientation of the head or the hands, that’s why will try to obtain them using inertial sensors. We will use a combination of gyroscopes, accelerometers and magnetometers to get the attitude of an object (in our case a human head). Each sensor cannot be used independently due to their high noise factor or their drift. We are going to first explain how the complementary Kalman filter works and how is used in inertial navigation. Next the sensor error models will be derived and finally an implementation and initialization of the filter will be developed. The state of our system will be described as \(x = [q, b_g, b_a]\) with its associated error state vector \(\delta x = [\delta \theta, \delta b_g, \delta b_a]\) that we will use in the filtering.

#### 6.1 Indirect complementary filter

Instead of using the full state of the system to do the estimation, we will work with the error state of the system and correct the original state at each iteration with the estimated error. The system is continuously integrated according to its dynamics and control inputs and a feedback correction term is added at every step proportional to the estimated error.

\[
\hat{x}_{t+1}^+ = \hat{x}_t^- + \delta \hat{x}_{t+1}^+ = \hat{x}_t^- + K_{++}\hat{z}_{t+1}
\] (6.1)

After every correction step the error is set to 0. Not doing so would mean taking into account two times the same error in the next correction step. The expected value of \(x_{t+1}^+\) after taking into account the sensor information is found by (5.1) making the following true

\[
\delta \hat{x}_{t+1}^+ = \hat{x}_{t+1}^- - E[x_{t+1}^+] = 0
\] (6.2)
Figure 5.4: Results with real data for the UKF top torso filter
This approach lets us integrate sensors like a gyroscope to drive the system and use a lower frequency sensor like an accelerometer, magnetometer or GPS to periodically correct the state. Working with the error state lets us think about the sensors variances in a more intuitive way. Having the complementary form allows us to make the filter more robust to sensor failures, if we detect a bad sensor reading we can skip the error correction and just continue integrating the system. Figures 6.1 and 6.2 show graphically the differences between the original KF and the newly proposed modification. The filter is summarized in table 6.1

6.2 Sensor error model

All the sensors that we are going to use are MEMS (Micro Electromechanical systems). None of them give perfect measurements and that is why we will derive their error models in order to compensate for their errors.

6.2.1 Gyroscope

A gyroscope gives the rotational velocity of the system. Integrating them we can get the new attitude of the system at every step but due to errors the integration will make the system drift. The measured gyroscope signal $w_m$ is corrupted by a
Indirect Kalman Filter

1. $\delta x_{t+1}^- = \Phi_t \delta x_t + w_t$
2. $\Sigma_{t+1}^- = \Phi_t \Sigma_t \Phi_t^T + R_t$
3. $z_{t+1} = \tilde{y} - h(x_{t+1})$
4. $K_{t+1} = \Sigma_{t+1}^- H_{t+1}^T (Q_{t+1} + H_{t+1} \Sigma_{t+1}^- H_{t+1}^T)^{-1}$
5. $\delta x_{t+1}^+ = \delta x_{t+1}^- + K_{t+1} (z_{t+1} - H \delta x_{t+1}^-)$
6. $\Sigma_{t+1}^+ = (I - K_{t+1} H_{t+1}) \Sigma_{t+1}^-$

State propagation

7. $x_{t+1}^- = f(x_t) + w_t$

State correction

8. $x_{t+1}^+ = x_{t+1}^- + \delta x_{t+1}^+$

Table 6.1: Indirect Kalman Filter
bias term \( b_g \) and noise rate term \( n_g \) that has Gaussian noise characteristics. The output of the sensor can be modelled as

\[
\omega_m = \omega + \Delta b_g + n_g
\]  

(6.3)

To correct the sensor errors we will have a estimate of the real angular rate of the sensor defined as

\[
\dot{\omega} = \omega_m - \Delta b_g
\]  

(6.4)

The following assumptions can be made

\[
\begin{align*}
E[n_g] &= 0_{3x1} \\
E[n_g \cdot n_g^T] &= I_{3,3}\sigma^2
\end{align*}
\]

(6.5)

and the bias term behaves as random walk process (Brownian noise)

\[
b_g = n_{bg}
\]

(6.6)

with characteristics

\[
\begin{align*}
E[n_{bg}] &= 0_{3x1} \\
E[n_{bg} \cdot n_{bg}^T] &= I_{3,3}\sigma^2
\end{align*}
\]

(6.7)

### 6.2.2 Accelerometer

The accelerometer gives the acceleration of the system. It has similar characteristics of the gyroscopes and we will use it to get the pitch and roll of the system measuring the gravity vector

\[
a_m = a + \Delta b_a + n_a
\]

(6.8)

The following assumptions can be made

\[
\begin{align*}
E[n_r] &= 0_{3x1} \\
E[n_r \cdot n_r^T] &= I_{3,3}\sigma^2
\end{align*}
\]

(6.9)

and the bias term behaves as random walk process (Brownian noise)

\[
b_a = n_{ag}
\]

(6.10)

with characteristics

\[
\begin{align*}
E[n_{ag}] &= 0_{3x1} \\
E[n_{ag} \cdot n_{ag}^T] &= I_{3,3}\sigma^2
\end{align*}
\]

(6.11)
6.2.3 Magnetometer

Since with the accelerometers the yaw is not observable we need and extra sensor, in this case the magnetometer. This sensor measures the direction of the magnetic north of the earth. It is very sensible to magnetic disturbances near the sensors that can be caused by other electronics or metal components. The hard iron bias is the combined result of the permanent magnetic elements near the sensor structure. The soft iron bias is the effect of ferromagnetic elements that affect the sensor when is in interaction with elements. We can correct the first type since it introduces a constant disturbance but cannot correct the second type since change continuously during the operation of the sensor depending on the environment. To do this we have to find the three offsets of the original signal, the three scale factors for the normalization of the axis and the three non-orthogonality angles which build an orthogonal system inside the sensors frame.

The sensor output is modelled as

\[ \mathbf{m}_m = \mathbf{U} \mathbf{m} + \mathbf{c}_m + \mathbf{n}_m \]  \hfill (6.12)

In order to calibrate the magnetometer we need to find the value of $\mathbf{U}$ and $\mathbf{c}_m$. From the geometric point of view originally all the measurements of the magnetometer should be in the surface of a three dimensional sphere of radius one. Due to hard iron elements the sphere has turned into a ellipsoid displace from its center, just as in figure ?? We will fit (6.13) and recover the calibrated measure as in figure ?? using

\[ (\mathbf{c} - \mathbf{v})^\top (\mathbf{U} \mathbf{U}^\top)(\mathbf{v} - \mathbf{c}) = 1 \]  \hfill (6.13)

\[ \mathbf{w} = \mathbf{U}(\mathbf{v} - \mathbf{c}) \]  \hfill (6.14)
6.3 Error quaternion

To use the complementary KF we need to derive the equations that drive the error state starting with the definition of the error quaternion and its derivative

\[ q = \delta q \otimes \dot{q} \]  
\[ \delta q = q \otimes \dot{q}^{-1} \]  
\[ \dot{q} = \delta \dot{q} \otimes \dot{q} + \delta q \otimes \dot{\dot{q}} \]

Substituting the definitions for \( \dot{q} \) and \( \dot{\dot{q}} \) using (6.15) leads to

\[ \frac{1}{2} \begin{bmatrix} \omega \\ 0 \end{bmatrix} \otimes q = \delta q \otimes \dot{q} + \delta q \otimes \left( \frac{1}{2} \begin{bmatrix} \omega \\ 0 \end{bmatrix} \otimes \dot{q} \right) \]

\[ \delta \dot{q} = \frac{1}{2} \begin{bmatrix} \omega \\ 0 \end{bmatrix} \otimes \delta q - \delta q \otimes \begin{bmatrix} \omega \\ 0 \end{bmatrix} \otimes \dot{q} \]  

We get the last line by multiplying the second line by \( \otimes \dot{q}^{-1} \) and using the expression (6.15). Since we also have to include the error dynamics of the gyroscope bias we include the gyroscope model defined in (6.3) and gyroscope estimate (6.4).
\[ \delta q = \frac{1}{2} \left( \begin{bmatrix} \omega \\ 0 \end{bmatrix} \otimes \delta q - \delta q \otimes \begin{bmatrix} \omega \\ 0 \end{bmatrix} \right) - \frac{1}{2} \begin{bmatrix} \Delta b_g + n_g \\ 0 \end{bmatrix} \otimes \delta q \] (6.19)

\[ = \frac{1}{2} \left( \begin{bmatrix} -[\omega \times] & 0 \\ -\omega^T & 0 \end{bmatrix} \delta q - \begin{bmatrix} +[\omega \times] & 0 \\ -\omega^T & 0 \end{bmatrix} \delta q \right) - \frac{1}{2} \begin{bmatrix} \Delta b_g + n_g \\ 0 \end{bmatrix} \otimes \delta q \] (6.20)

\[ = \frac{1}{2} \left( \begin{bmatrix} -[\omega \times] & 0_{3 \times 1} \\ -0_{3 \times 1} & 0 \end{bmatrix} \delta q - \frac{1}{2} \begin{bmatrix} \Delta b_g + n_g \\ 0 \end{bmatrix} \otimes \delta q \right) \] (6.21)

\[ = \frac{1}{2} \left( \begin{bmatrix} -[\omega \times] & 0_{3 \times 1} \\ -0_{3 \times 1} & 0 \end{bmatrix} \delta q - \frac{1}{2} \begin{bmatrix} -[\Delta b_g + n_g \times] & \Delta b + n_r \end{bmatrix} \begin{bmatrix} \delta q \\ 1 \end{bmatrix} \right) \] (6.22)

\[ = \frac{1}{2} \left( \begin{bmatrix} -[\omega \times] & 0_{3 \times 1} \\ -0_{3 \times 1} & 0 \end{bmatrix} \delta q - \frac{1}{2} \begin{bmatrix} (\Delta b_g + n_g) \end{bmatrix} \right) + \text{h.o.t.'s} \] (6.23)

Where from (6.22) to (6.23) we have neglected all higher order terms that come up when multiplying the two error factors.

The final system is defined as

\[ \delta \dot{q} = \begin{bmatrix} \delta \dot{q} \\ \delta \dot{q} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \dot{\delta \theta} \\ 1 \end{bmatrix} = \begin{bmatrix} -\hat{w} \times \delta q - \frac{1}{2}(\Delta b + n_r) \end{bmatrix} \] (6.24)

which can be also stated as

\[ \delta \dot{\theta} = -\hat{w} \times \delta \theta - \Delta b - n_r \] (6.25)

The equation that governs both bias errors is

\[ \Delta b = \dot{b} - \hat{b} = n_w \] (6.26)

Combining both we get the the system that describes the error dynamics in the form \( \dot{x} = Fx + Gn \)

\[ \begin{bmatrix} \dot{\delta \theta} \\ \Delta \dot{b} \\ \Delta \dot{b}_w \end{bmatrix} = \begin{bmatrix} -[\omega \times] & -I_{3 \times 3} & 0_{3 \times 3} \\ 0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} \end{bmatrix} \begin{bmatrix} \delta \theta \\ \Delta b \\ \Delta b_w \end{bmatrix} + \begin{bmatrix} -I_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} \\ 0_{3 \times 3} & I_{3 \times 3} & 0_{3 \times 3} \end{bmatrix} \begin{bmatrix} n_r \\ n_w \end{bmatrix} \] (6.27)

Because the system is linear with respect to the state vector we can also obtain the solution to the differential as in (6.8) using the matrix exponential. The resulting discrete state transition matrix is

\[ \Phi(t_{k+1}, t_k) = \begin{bmatrix} \Theta & \Psi & 0_{3 \times 3} \\ 0_{3 \times 3} & I_{3 \times 3} & 0_{3 \times 3} \end{bmatrix} \] (6.28)
Where each of the $\Theta$ and $\Psi$ are defined as

$$\Theta = I_{3x3} - \Delta t [w \times] + \frac{\Delta t^2}{2} [w \times]^2$$ (6.29)

$$\Psi = -I_{3x3} \Delta t + \frac{\Delta t^2}{2} [w \times] - \frac{\Delta t^3}{6} [w \times]^2$$ (6.30)

### 6.4 Measurement model

We have two aiding sensors, which will be used to correct the drift and precision of the system. In both cases we know the value of the output of the sensor if it is aligned with the inertial frame that we are working, in the case of the accelerometer is the gravity vector $g = [0 \ 0 \ -9.8]$ and in the magnetometer that points to the magnetic north of the earth $n = [1 \ 0 \ 0]$. Given the actual output of the sensors we can infer what change in rotation with respect to the inertial frame orientation has been made to the sensors. As we said they are noisy estimates, specially for the magnetometer since it not only reads the earth’s gravity but also the body’s acceleration (we will treat this extra term as noise), and the magnetometer is very sensible to magnetic disturbances.

First we will the derive the general case for a measurement and after the individual models for each of them. The relationship that we described can be expressed as

$$L_r = \hat{L}_G C^G r$$ (6.31)

the actual measurement of the sensor its a projection of vector $r$ in the sensors reference frame corrupted by noise and added changing bias driven by a random walk

$$z = \hat{L}_G C^G r + n_m + b_m$$ (6.32)

For the update of the indirect Kalman filter we need to relate the measurement error $z$ to the state vector $\delta x$

$$z = \tilde{y} - h(\hat{x}) = (\hat{L}_G C(q) - \hat{\dot{L}} G C(\hat{q}))^G r + n_m + b_m$$ (6.33)

We first define

$$\hat{L}_G C(q) - \hat{\dot{L}} G C(\hat{q}) = (\hat{L}_G C(q) - I_{3x3})^L G C(\hat{q}) = -[\delta \theta \times]_G C(\hat{q})$$ (6.34)
where we have used the relation (2.2.1) to get $\delta \mathbf{x}$ into the equation. This will be used to the error vector into the measurement equation. Now we can write the equation (6.33) as

$$z = (\mathbf{L} C(\mathbf{q}) - \mathbf{I}_{3\times3}) \mathbf{C}(\mathbf{q})^G \mathbf{r} + \mathbf{n}_m + \mathbf{b}_m$$

$$= (\mathbf{L} C(\mathbf{q}) - \mathbf{L} C(\mathbf{q})^G \mathbf{r} + \mathbf{n}_m + \mathbf{b}_m$$

$$= (\mathbf{L} C(\mathbf{q})^G \mathbf{r} + \mathbf{n}_m + \mathbf{b}_m$$

$$= (\mathbf{L} C(\mathbf{q}))^G \mathbf{r} \times [\delta \theta + \mathbf{n}_m + \mathbf{b}_m$$

$$= \begin{bmatrix} \mathbf{L} C(\mathbf{q})^G \mathbf{r} \times \\ \delta \theta \end{bmatrix} \begin{bmatrix} \delta \theta \\ \mathbf{b}_w \end{bmatrix} + \mathbf{n}_m + \mathbf{b}_m$$

(6.35)

Now that we have the model for the general measurement we will derive the particular model for the accelerometer and magnetometer taking into account the full state of the system.

6.4.1 Accelerometer measurement model

The vector that we are measuring is the gravity $\mathbf{g}$. The sensors output has added Gaussian noise and a bias.

$$z_a = \left[ \begin{bmatrix} \mathbf{L} C(\mathbf{q})^G \mathbf{g} \times \\ \delta \theta \end{bmatrix} \begin{bmatrix} \delta \theta \\ \mathbf{b}_w \end{bmatrix} + \mathbf{n}_a + \mu(t) \right.$$

(6.36)

where $\mu(t)$ is the acceleration of the system that varies with time. Since we cannot observe when the system is acceleration we will treat this term also as extra noise. When ever the norm of the acceleration sensor differs from the norm of the gravity vector, in our case 9.8 we will increase proportionally the variance of the sensor reading, meaning that the filter will ignore the measurement as long as the system is accelerating.

6.4.2 Magnetometer measuring model

We will assume that the reading of the magnetometer has already been calibrated to output the value $\mathbf{m} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ when its pointing to the magnetic north, by means of section 6.2.3.

The model for the magnetometer is

$$z_a = \left[ \begin{bmatrix} \mathbf{L} C(\mathbf{q})^G \mathbf{g} \times \\ \delta \theta \end{bmatrix} \begin{bmatrix} \delta \theta \\ \mathbf{b}_w \end{bmatrix} + \mathbf{n}_a + \mu(t) \right.$$
$$\mathbf{z}_m = \begin{bmatrix} [\hat{L}_G C(\hat{q})]^G \mathbf{m} \times \end{bmatrix} 0_{3 \times 3} 0_{3 \times 3} \begin{bmatrix} \delta \theta \\ \delta \phi \\ \delta \psi \end{bmatrix} + \mathbf{n}_m$$  

(6.37)

6.5 Initialization

We are going to find an estimate of the rotation $\mathbf{C}$ from the inertial frame to the body looking at the aiding sensors and use this matrix as an initialization point.

Given the fact that the magnetic vector $\mathbf{m}$ and accelerometer measure $\mathbf{a}$ are in different directions we can create an initialization routine with the extra vector $\mathbf{r} = \mathbf{g} \times \mathbf{m}$ that is orthogonal to both $\mathbf{g}$ and $\mathbf{m}$. We can create the following matrix in the inertial frame

$$^G \mathbf{A} = \begin{bmatrix} ^G \mathbf{m} & ^G \mathbf{g} \times ^G \mathbf{m} & ^G \mathbf{g} \end{bmatrix} = \begin{bmatrix} m_e & 0 & 0 \\ 0 & m_e g_e & 0 \\ 0 & 0 & g_e \end{bmatrix}$$  

(6.38)

with the known quantities $m_e$ equal to the earth magnetic field and $g_e$ the gravity of the earth. Because $^L \mathbf{m} = ^L \mathbf{R}^G \mathbf{m}$, $^L \mathbf{r} = ^L \mathbf{R}^G \mathbf{r}$, $^L \mathbf{r} = ^L \mathbf{C}^G \mathbf{r}$, $^L \mathbf{g} = ^L \mathbf{R}^G \mathbf{g}$ we have that

$$^L \mathbf{A} = ^L \mathbf{R}^G \mathbf{A}$$  

(6.39)

where

$$^L \mathbf{A} = \begin{bmatrix} ^L \mathbf{m} & ^L \mathbf{g} \times ^L \mathbf{m} & ^L \mathbf{g} \end{bmatrix}$$  

(6.40)

with $^G \mathbf{A}$ and $^L \mathbf{A}$ known we can find the value of $\mathbf{C}$ as

$$\hat{\mathbf{R}} = ^L \hat{\mathbf{A}} (^G \mathbf{A})^{-1} = \begin{bmatrix} \frac{1}{m_e} ^L \mathbf{m} & \frac{1}{m_e m_g} ^L \mathbf{g} \times ^L \mathbf{m} & \frac{1}{g_e} ^L \mathbf{g} \end{bmatrix}$$  

(6.41)

6.6 Filter implementation and results

The filter sequence is:

1. Create a new measurement of $\mathbf{w}_t$ from $\mathbf{w}_m$ using $b_t$.

2. Propagate the orientation using (2.20) and the new estimate $\hat{\mathbf{w}}_t$.

3. Compute the measurement matrix composed of (4.1) concatenated vertically.

4. Compute the measurement residuals $z_a, z_m$ and concatenate them vertically in a vector.
5. Perform the KF described in table 6.1.

6. Compute the quaternion correction

\[
\delta \hat{q}_c = \begin{bmatrix} \delta q \\ \sqrt{1 - \delta q^\top \delta q} \end{bmatrix} 
\]

(6.42)

or if \(\delta q^\top \delta q > 1\) using

\[
\delta \hat{q}_c = \begin{bmatrix} \delta q \\ \sqrt{1 + \delta q^\top \delta q} \end{bmatrix} \begin{bmatrix} \delta q \\ 1 \end{bmatrix}
\]

(6.43)

7. Correct the actual orientation using (2.15) with the calculated quaternion correction \(\delta \hat{q}_c\). Update both the bias with \(\delta b\)

\[
\hat{q}_{t+1}^+ = \delta \hat{q}_c \otimes \hat{q}_{t+1}^- \\
\hat{b}_{t+1}^+ = \Delta b + \hat{b}
\]

(6.44)

8. Set the error state \(\delta x = 0\)

### 6.6.1 Simulation

To validate our design we will generate a trajectory shown in 6.4. The trajectory is the parametrized curve (4.9). Because we have a parametrized curve we can get the acceleration of the body computing the second derivative. The simulated acceleration measure will sense the acceleration of the moving reference frame plus the gravity in its own frame, which also rotates at angular velocity equal to the \(z\) component of the curve (6.45). The same thing applies to the simulated magnetometer. In the simulation we have added noise with \(\sigma = 0.1\) to the gyroscope and a constant bias of 0.1. The accelerometer and magnetometer also have also an added noise of \(\sigma = 0.1\)

\[
x = \cos(t)(2 - \cos(\frac{2t}{3})) \\
y = \sin(t)(2 - \cos(\frac{2t}{3})) \\
z = -\sin(\frac{2t}{3})
\]

(6.45)

Figure 6.5 shows the result of the filtering compared with the real values of the frames orientation parametrized by a quaternion and the orientation that we would
get if we didn’t filter and only looked at the aiding sensors using the same technique that we used for the initialization. Figure 6.4 shows in contrast what would happen if we only integrated the gyroscope signal to get the orientation without using of the other sensors. It can be seen how the estimate quickly drift to an erroneous solution.

![Figure 6.4: AHRS simulated trajectory](image)

### 6.6.2 Real data

Here we show the results of the filter with real data. Figure 6.6.2 show a picture of IMU used, and figure 7.1 show the results of the filter. A sensor network was made using Zigbees to connect each IMU to the computer that processes the data.

### 6.7 Robot wrist teleoperation

The first use case of the teleoperation only uses position in the inverse kinematics but now we also have orientation constraints. Only using the users hands position and orientation mapped to the robot is not a good solution since small rotation of the users hands create large accelerations on the full robot coupling all the movement of the robot arms. The way to circumvent this is by trying this time to specify as much as possible the inverse kinematic solution. This is done by specifying in the first task the position and orientation of the hands, elbows and head all at the same time. Our problem has become once again a least squares estimation problem.
Figure 6.5: AHRS with the simulated trajectory

problem since the problem is now overdetermined. Figure 6.9 shows the head added to the teleoperation and figure 6.10 shows the robot following the orientation of both hands of the users.
Figure 6.6: AHRS with only gyro integrated

Figure 6.7: AHRS with real data
Figure 6.8: AHRS with real data
Figure 6.9: Results of the teleoperation using the head
Figure 6.10: Results of the teleoperation using the hands
7 Conclusion

Many subjects have treated during this work and one of the main goals was to show how all of them can be connected. This thesis has successfully showed how optimal filtering can be applied to solve different problems. Two applications come out of this work. The first is the teleoperation of a humanoid robot using different inverse kinematics schemes. The second applications is an attitude and reference system based on mems sensors that is also used in the robots teleoperation.

7.1 Time planification

7.2 Costs & Resources

This project has needed a material investment of 548 €. The details are in table 7.1. A time investment of five months with an average of eight hours a day gives 800 hours of work. Each hour is paid at fifteen € giving a total of 12000 € for the human investment in the project. The final cost is 12548 €. Not using a commercial IMU has dropped the investment price by a factor of five and with the developed work the company can further reduce the price by making custom IMU boards since all the software and firmware code has been developed. The human investment has been justified with the product that has come out of it, a way to teleoperate a humanoid robot with the maximum expressiveness. This product can be used to teach the robot movements in a very fast and intuitive way and can also be used as a marketing claim do to the visual impact that the teleoperation gives.
<table>
<thead>
<tr>
<th>Material</th>
<th>Quantity</th>
<th>Price</th>
<th>Subtotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Razor IMU</td>
<td>3</td>
<td>100</td>
<td>300</td>
</tr>
<tr>
<td>zigbee</td>
<td>4</td>
<td>17</td>
<td>68</td>
</tr>
<tr>
<td>IMU daughter board</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>IMU programming board</td>
<td>1</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Lipo batteries</td>
<td>3</td>
<td>15</td>
<td>45</td>
</tr>
<tr>
<td>Kinect</td>
<td>1</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>548 €</strong></td>
</tr>
</tbody>
</table>

Table 7.1: Materials cost
References


