Some Essential Notes for Roboticists

Andreu Corominas-Murtra *

October 26, 2020

Abstract

This document is a running collection of math notes. They try to summarize in a clear and useful way the math essentials for advanced robotics. The focus is put more on the interpretation of math entities and results, instead on computation and numerical recipes. The aim is to build a self reference tool for our daily work, and to share it.

*Andreu Corominas-Murtra (andreu@beta-robots.com) is with Beta Robots (www.beta-robots.com) In case of reference, please cite this as: Corominas-Murtra, A. Some Essential Notes for Roboticists. On-line at `https://github.com/andreucm/essential_maths_roboticists`. 2019.

Contents

Notation

Main conventions and notation used through the document are listed below:

$a, b, c, x, y, z, u, v, w, ...$ are scalars.
$\alpha, \beta, \theta, \phi$ are also scalars, by they usually represent angles.
$\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{o}, \mathbf{p}, \mathbf{r}, \mathbf{u}, \mathbf{v}, \mathbf{w}, ...$ are vectors.
$\mathbf{M}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{F}, ...$ are matrices.
$\mathbf{R}$ is a rotation matrix.
$\mathbf{T}$ is a homogeneous transform matrix.
$\mathbf{I}_n$ is the identity matrix of dimensions $n \times n$.
$\mathscr{F}^C$ is the coordinate frame (shortly, frame) of body C.
$\mathbf{x}^C$ is a vector referenced with respect to the $\mathscr{F}^C$ coordinate frame.
$\mathbf{a} \in \mathbb{R}^3 \rightarrow \mathbf{a} = (a_x, a_y, a_z)^T$ are the components of a vector in 3D.
$\mathbf{a}_x^C$ is the first component of $\mathbf{a}$ expressed in frame $\mathscr{F}^C$.
$i, j, k, l, .. \in \mathbb{Z}$ are usually running index over sets.
$t \in \mathbb{Z}$ is usually an iteration index.
$\tau \in \mathbb{R}$ is usually a timestamp.
$\tau_t \in \mathbb{R}$ is the timestamp of iteration t.
$m, n, p, .. \in \mathbb{Z}$ are usually vector or matrix dimensions.
$p \in \mathbb{Z}$ is also used as a degree index of some function or polynomial.
$\mathbf{q} = (q_r, q_x, q_y, q_z) = (a, b, c, d)$ is a quaternion. q_r or a are the real parts.
$\hat{\mathbf{x}}$ is an estimate of $\mathbf{x}$, which is an actual (unknown) quantity.

Part I
Math essentials

1 Basic Trigonometry

Trigonometry provides tools on the relations between angles and distance ratios in right triangles. Given the circle of unit radius ($r = 1$) in figure 1, the cosine, sine and tangent are defined as:

$$\cos\alpha = \frac{a}{r} = \frac{a}{1} = a; \quad \sin\alpha = \frac{b}{r} = \frac{b}{1} = b; \quad \tan\alpha = \frac{b}{a}; \tag{1}$$

The above functions are defined as $f(\alpha) : \mathbb{R} \to \mathbb{R}$. They are periodic functions with period 2π, meaning that the function provides the same value for an argument $\alpha + k2\pi$, being k an integer ($k \in \mathbb{Z}$).

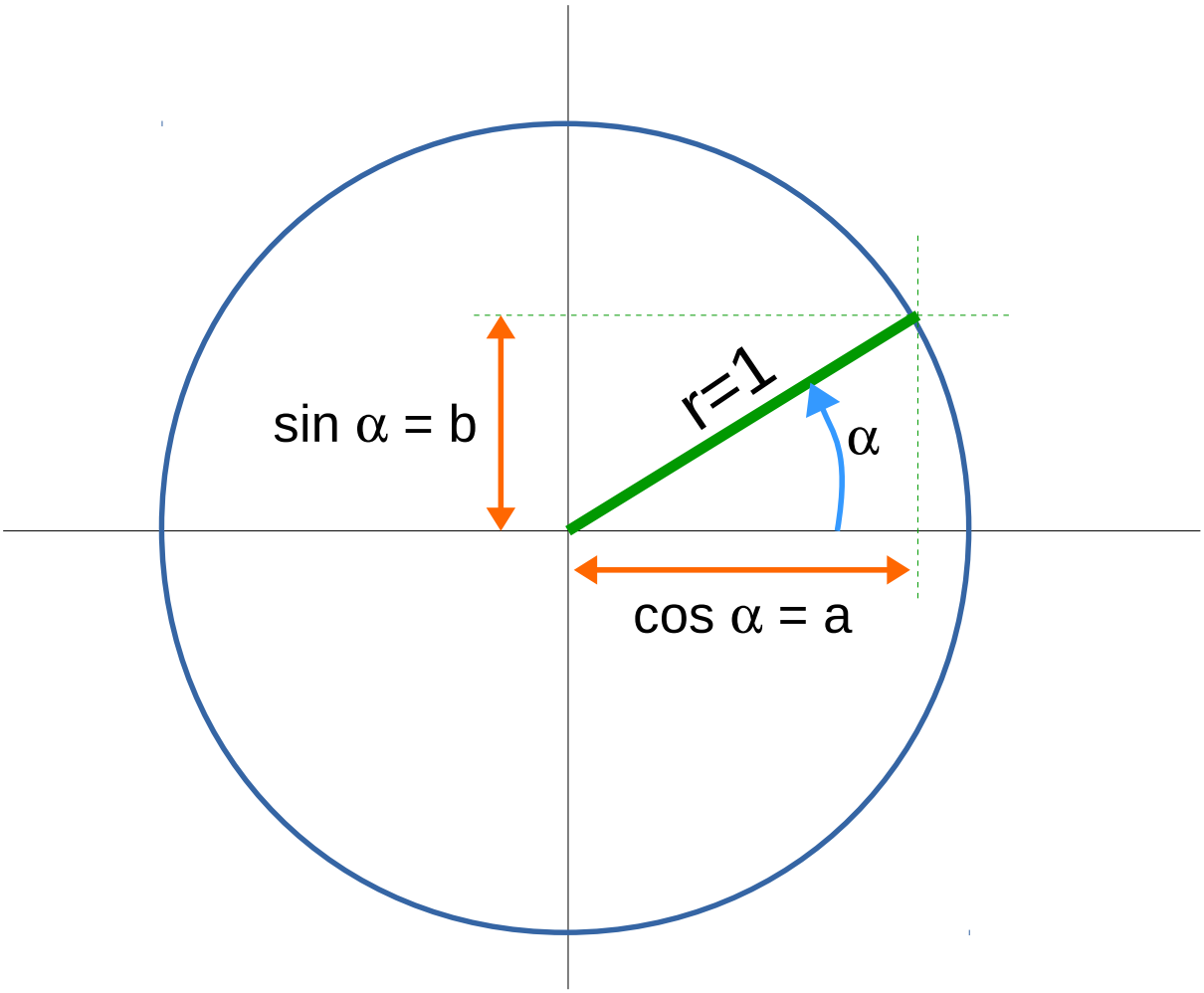

Figure 1: Unit circle and deifinitions of cos, sin and tan. (to do)

The inverse functions $acos()$, $asin()$ and $atan()$ provides an angle given a ratio of distances.

2 Basic Algebra and Calculus

2.1 Functions

A real **scalar function** defined in the $\mathbb{R}^n$ domain is:

$$z = f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R},\ \ z \in \mathbb{R},\ \mathbf{x} \in \mathbb{R}^n. \tag{2}$$

A real **vector function** defined in the $\mathbb{R}^n$ domain is:

$$\mathbf{z} = f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m,\ \ \mathbf{z} \in \mathbb{R}^m,\ \mathbf{x} \in \mathbb{R}^n. \tag{3}$$

2.2 Norm, dot and cross products

The **scalar or dot product** between two vectors is:

$$< \mathbf{u}, \mathbf{v} >= \mathbf{u}^T \cdot \mathbf{v} = \mathbf{u}^T \mathbf{v} = \sum_{i=0}^{n-1} u_i v_i,\ \ \mathbf{u}, \mathbf{v} \in \mathbb{R}^n. \tag{4}$$

The scalar product is a measure of the length of the projection of one vector over the axis of the other. Therefore, the scalar product can be also interpreted as a measure of similarity.

The p-**norm** of a vector is

$$|\mathbf{x}|_p = (\sum_{i=0}^{n-1} x_i^p)^{\frac{1}{p}}\ . \tag{5}$$

For $p = 2$, we found the **Euclidean norm**:

$$|\mathbf{x}|_2 = |\mathbf{x}| = \sqrt{\sum_{i=0}^{n-1} x_i^2} = \sqrt{\mathbf{x}^T \mathbf{x}}\ , \tag{6}$$

which is a measure of the length of $\mathbf{x}$, a fundamental concept.

If a vector is divided by its norm, $\frac{\mathbf{x}}{|\mathbf{x}|}$, we obtain a **normalized** version, which can be interpreted as a vector of unit length pointing to the same direction of $\mathbf{x}$. Thus, normalized vectors indicate directions and lie on the hypersphere of unit radius. In 3D this can be interpreted as an **spherical projection**.

The scalar product between normalized $\mathbf{u}$ and $\mathbf{v}$ equals to the cosine of the angle between them:

$$\frac{\mathbf{u}^T}{|\mathbf{u}|} \cdot \frac{\mathbf{v}}{|\mathbf{v}|} = \frac{\mathbf{u}^T \mathbf{v}}{|\mathbf{u}||\mathbf{v}|} = cos(\alpha),\ \in [0,1] \tag{7}$$

where α is the angle between vectors $\mathbf{u}$ and $\mathbf{v}$. This operation can be also interpreted as a similarity value between two normalized vectors. If normalized vectors are equal, the angle is 0, so the cosine is 1 (maximum similarity). Otherwise, if vectors are orthogonal, the angle is $\frac{\pi}{2}$, so the cosine results in 0 (minimum similarity). In the later case $\mathbf{u}$ and $\mathbf{v}$ span orthogonal directions.

The **cross product** between two vectors is only defined in $\mathbb{R}^3$

$$\mathbf{u} \times \mathbf{v} = \mathbf{w},\ \ \mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{R}^3, \tag{8}$$

and results in a new vector $\mathbf{w}$ which is orthogonal to both $\mathbf{u}$ and $\mathbf{v}$. Components of the resulting vector are computed as:

$$\mathbf{w} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \times \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} u_y v_z - u_z v_y \\ u_z v_x - u_x v_z \\ u_x v_y - u_y v_x \end{bmatrix} \tag{9}$$

Right hand rule, indicates forward direction of the resulting vector. That is, using the right hand, forefinger pointing as u, middle finger pointing following v, so the resulting w is orthogonal to both, and points as thumb.

Example 1. Code for norm, dot and cross products [C++/Eigen]

```
#include eigen3/Eigen/Geometry
 void main()
 {
    Eigen::Vector3d va; //va is a vector in R^3
    Eigen::Vector<double,3> vb; //vb is another vector in R^3
    va << 1,2,3; //fill the vector va
    vb << -1,-1,0; //fill the vector vb
    double norm_va = va.norm(); //|va|
    double dot_vab = va.dot(vb); //va^T · vb
    Eigen::Vector3d cross_vab = va.cross(vb);//va x vb
 }
```

2.3 Matrix Manipulation

A **matrix** is a rectangular array of numbers, so they are sorted in rows and columns:

$$\mathbf{A} \in \mathbb{R}^{m \times n} \begin{bmatrix} a_{00} & \dots & a_{0j} & \dots & a_{0n-1} \\ a_{10} & \dots & \dots & \dots & a_{1n-1} \\ \dots & \dots & \dots & \dots & \dots \\ a_{i0} & \dots & a_{ij} & \dots & a_{in-1} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m-10} & \dots & a_{m-1j} & \dots & a_{m-1n-1} \end{bmatrix} \tag{10}$$

A matrix lies on a $\mathbb{R}^{m \times n}$ space, where the dimensions are the number of rows m, and the number of columns n. Element a_{ij} represents a generic element of the matrix $\mathbf{A}$. Matrices are usually interpreted as a set of column or row vectors: a set of m row vectors of dimension n, or a set of n column vectors of dimension m. Following this interpretation, it can be written:

$$\mathbf{A} = \begin{bmatrix} \bar{\mathbf{a}}_0 \\ \bar{\mathbf{a}}_1 \\ \dots \\ \bar{\mathbf{a}}_i \\ \dots \\ \bar{\mathbf{a}}_{m-1} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_0 & \dots & \mathbf{a}_j & \dots & \mathbf{a}_{n-1} \end{bmatrix}, \tag{11}$$

where $\bar{\mathbf{a}}_i$ means the row vector corresponding to row i.

A matrix is called **squared** when $n = m$.

A matrix is called **diagonal** when $a_{ij} = 0, \forall i \neq j$, and $a_{ij} \neq 0, \forall i = j$

A matrix is called **orthogonal** when its rows or columns are orthogonal vectors, which can be checked if their scalar product is zero.

A matrix is called **upper triangular** when $a_{ij} = 0, \forall i > j$, and **lower triangular** when $a_{ij} = 0, \forall i < j$.

$\mathbf{I}_n$ is the identity matrix, which is a diagonal matrix with all non-zero elements set to 1.

The **sum** of two matrix is done element by element, so both matrix should have the same dimensions, as well as the resulting matrix:

$$\mathbf{A} = \mathbf{B} + \mathbf{C},\ \mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{m \times n} \ \rightarrow a_{ij} = b_{ij} + c_{ij} \tag{12}$$

The matrix **product**

$$\mathbf{A} = \mathbf{B}\,\mathbf{C},\ \mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{B} \in \mathbb{R}^{m \times p}, \mathbf{C} \in \mathbb{R}^{p \times n} \tag{13}$$

is computed as follows:

$$a_{ij} = \bar{\mathbf{b}}_i \cdot \mathbf{c}_j = \sum_{i=0}^{p-1} b_{ip} c_{pj}. \tag{14}$$

Each resulting component can be viewed as a scalar product between $\bar{\mathbf{b}}_i$ and $\mathbf{c}_j$, so all comments exposed above regarding the scalar product can be interpreted for the matrix product. Please note in equation 13 matrix dimensions of **B** and **C**, and the resulting dimensions of **A**. Matrix multiplication is allowed if and only if the number of columns of the first operand is equal to the number of rows of the second operand. Therefore, matrix multiplication is not a commutative operation in a general case.

The **rank** of a matrix is the minimum between the number of independent rows and the number of independent columns. An independent vector (row or column) means that it cannot be expressed as a linear combination of the other vectors (rows or columns) of the matrix. Therefore an independent vector adds a new dimension to the space the vectors of the matrix are spanning.

The **matrix determinant** is defined for squared matrices, and it is computed following this recursive expression:

$$det(\mathbf{A}) = |\mathbf{A}| = \sum_{j=0..n-1} (-1)^{1+j} \, a_{ij} \, det(\mathbf{A}_{-0j}) \tag{15}$$

where $\mathbf{A}_{-0j}$ is the matrix composed by **A** but removing its first row and column j. For $n = 2$ this expression is reduced to:

$$\begin{vmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{vmatrix} = a_{00}a_{11} - a_{01}a_{10} \tag{16}$$

A matrix is called rank deficient or null-rank when its determinant is zero, meaning that some row or column can be expressed as a linear combination of other rows or columns respectively.

The determinant is a measure of the hypervolume of the hyperellipsoide represented by matrix **A**.

The **transpose** of **A**, only defined for squared matrices, is $\mathbf{A}^T$, where its elements exchange columns by rows:

$$\mathbf{A} = \{a_{ij}\}; \; \mathbf{A}^T = \{a_{ji}\} \tag{17}$$

Given a squared matrix **A** of sizes $n \times n$, $\mathbf{A}^{-1}$ is the **inverse** of a **A**, so it fulfills that $\mathbf{A}^{-1}$ is also squared $n \times n$ and:

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}_n \tag{18}$$

The following is a list of useful expressions [1], [...]:

$$(\mathbf{A}^{-1})^{-1} = \mathbf{A} \tag{19}$$
$$(\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T \tag{20}$$
$$(a\mathbf{A})^{-1} = a^{-1}\mathbf{A}^{-1} \tag{21}$$
$$(\mathbf{A}\mathbf{B})^T = \mathbf{B}^T\mathbf{A}^T \tag{22}$$
$$(\mathbf{A}\mathbf{B})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1} \tag{23}$$
$$\mathbf{A}(\mathbf{B}+\mathbf{C}) = \mathbf{A}\mathbf{B} + \mathbf{A}\mathbf{C} \tag{24}$$
$$(\mathbf{A}+\mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T \tag{25}$$
$$(\mathbf{A}+\mathbf{B})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{I}+\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{A}^{-1} \tag{26}$$

The following equation shows the **matrix inversion lemma**. It's a useful expression in situations where matrix **A** and **C** are easy to invert, because they may be diagonal or have small dimension. In such case it can be applied:

$$(\mathbf{A}+\mathbf{B}\mathbf{C}\mathbf{D})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}\mathbf{C}(\mathbf{C}+\mathbf{C}\mathbf{D}\mathbf{A}^{-1}\mathbf{B}\mathbf{C})^{-1}\mathbf{C}\mathbf{D}\mathbf{A}^{-1} \tag{27}$$

The prove can be found at[ref], but it starts by multiplying at each side by $(\mathbf{A}+\mathbf{BCD})$.

The **skew-symmetric matrix** in $\mathbb{R}^3$ is defined as a square matrix whose transpose is also its negative:

$$\mathbf{A} = \begin{bmatrix} 0 & a & -b \\ -a & 0 & c \\ b & -c & 0 \end{bmatrix}. \tag{28}$$

The skew matrix can be useful to represent the vector product as a matrix-vector product. If $\mathbf{a} = [a_x, a_y, a_z]$ is a vector in $\mathbb{R}^3$, let be $\mathbf{A}$ the following matrix:

$$\mathbf{A}_s(\mathbf{a}) = \mathbf{A}_s \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}, \tag{29}$$

then the vector product of $\mathbf{a} \times \mathbf{b}$ can be expressed as:

$$\mathbf{a} \times \mathbf{b} = \mathbf{A}_s \mathbf{b} \tag{30}$$

A **hermitian** matrix is that fulfilling:

- $\mathbf{A}$ is square $n \times n$.
- $\mathbf{A} = (\mathbf{A}^T)^* \rightarrow a_{ij} = a_{ji}^*$

And a **positive-definite** matrix is that fulfilling:

- $\mathbf{A}$ is square $n \times n$.
- $\mathbf{v}^T \mathbf{A} \mathbf{v} > 0$, $\forall \mathbf{v} \in \mathbb{R}^n$, and $\neq \mathbf{0}$.

Both later cases are commonly found in practical robotics applications and present interesting properties exploited by algebra programming libraries.

3 Rigid Transformations

Rigid transformations is a major topic in robotics. From perception, to kinematics and control, everywhere the concept of a rigid transformation appears. A rigid transformation defines the rotation and translation between two coordinate frames. How both this translation, and specially this rotation, are represented (or parameterized) introduce fundamental concepts reviewed in the following paragraphes.

Coordinate frames will be written as $\mathscr{F}^A$, $\mathscr{F}^B$, for frames A and B. They could be positioned, for instance, at the pivot point of a platform and at the center point of a sensor.

3.1 Rotations

There are several ways to represent a rotation. The most used ones are the following: rotation matrix, quaternion, Euler angles and axis angle. Each representation has benefits and drawbacks, depending on the application. For instance, for a sea ship which will never flip (hopefully), Euler angles could be a good option, but for an end effector tool expected to cover all orientations in 3D, quaternions or Rotation matrix could be better.

3.1.1 Rotation Matrix

Definition A rotation matrix $\mathbf{R}$ is a squared, orthogonal matrix with $|\mathbf{R}| = 1$.

Interpretation Columns of $\mathbf{R}$ can be interpreted as orthonormal vectors expressing a frame $\mathscr{F}^B$ in terms of another frame $\mathscr{F}^A$. In that case, such rotation matrix is expressed as $\mathbf{R}_B^A$. Given two frames which are rotated, $\mathscr{F}^A$ and $\mathscr{F}^B$, and given a vector $\mathbf{v}^B$ expressed in terms of $\mathscr{F}^B$, the following equation expresses the vector $\mathbf{v}$ with its new coordinates in terms of $\mathscr{F}^A$:

$$\mathbf{v}^A = \mathbf{R}_B^A \mathbf{v}^B \tag{31}$$

It fulfills also

$$\mathbf{v}^B = (\mathbf{R}_B^A)^{-1} \mathbf{v}^A = \mathbf{R}_A^B \mathbf{v}^A \tag{32}$$

The number of parameters required for this parameterization is $n \times n$ in an n-dimension space. In 3D, for instance, 9 numbers are required. However, these 9 parameters have to fulfill certain conditions between them, so they are not free. These conditions are imposed by the definition of the rotation matrix itself: orthogonality and unit determinant.

Chain of Rotations If a third frame $\mathscr{F}^C$ is also rotated with respect to frame $\mathscr{F}^B$, then the equation representing the chain of rotations, from a point expressed in terms of $\mathscr{F}^C$ to the point expressed in terms of $\mathscr{F}^A$, is:

$$\mathbf{v}^A = \mathbf{R}_B^A \mathbf{R}_C^B \mathbf{v}^C = \mathbf{R}_C^A \mathbf{v}^C \tag{33}$$

Please be careful with the sequence of the matrix product representing the chain, since matrices are ordered from right to left, starting from the *outer* rotation up to the *inner* one. In the 3D case, computing $\mathbf{R}_C^A$ requires 27 multiplications and 18 additions.

Example 2. Rotations in 2D In 2D, if frame $\mathscr{F}^B$ is rotated by $\alpha = 30^o$ degrees with respect to frame $\mathscr{F}^A$, the matrix representing this rotation is as follows:

$$\mathbf{R}_B^A = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \tag{34}$$

Given $\mathbf{v}^B = [2\ 1]^T$ a vector (point) represented in terms of frame $\mathscr{F}^B$, its coordinates in terms of frame $\mathscr{F}^A$ are:

$$\mathbf{v}^A = \mathbf{R}_B^A \mathbf{v}^B = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.232 \\ 1.866 \end{bmatrix} \tag{35}$$

so $\mathbf{v}^A = [1.232\ 1.866]^T$ is the same point $\mathbf{v}$ but expressed in terms of frame $\mathscr{F}^A$.

3.1.2 Quaternions

Definition A quaternion[ref] is an extension of Complex numbers to 3D. As complex numbers can be represented as vectors in $\mathbb{R}^2$, quaternions can be represented as vectors in $\mathbb{R}^4$, with one real part, a, and three imaginary parts, b, c, d:

$$\mathbf{q} = [q_r\ q_i\ q_j\ q_k]^T = q_r\mathbf{1} + q_i\mathbf{i} + q_j\mathbf{j} + q_k\mathbf{k} = q_r + q_i\mathbf{i} + q_j\mathbf{j} + q_k\mathbf{k}, \tag{36}$$

so quaterions are defined over a base formed by vectors $\mathbf{1}, \mathbf{i}, \mathbf{j}, \mathbf{k}$. Definition of this base is required to define the quaternion product. Vectors of this base fulfill:

- $\mathbf{1}$ is the identity element $\rightarrow\ \mathbf{q1} = \mathbf{q}$
- $\mathbf{ii} = \mathbf{jj} = \mathbf{kk} = \mathbf{ijk} = -1$
- $\mathbf{ij} = \mathbf{k};\ \mathbf{jk} = \mathbf{i};\ \mathbf{ki} = \mathbf{j};$

The **sum** of two quaternions, $\mathbf{q}^a$ and $\mathbf{q}^b$, is defined as the sum in $\mathbb{R}^4$:

$$\mathbf{q} = \mathbf{q}^a + \mathbf{q}^b = [(q_r^a + q_r^b)\ (q_i^a + q_i^b)\ (q_j^a + q_j^b)\ (q_k^a + q_k^b)\ ]^T \tag{37}$$

and the **scalar product** of a quaternion is:

$$u\mathbf{q} = [uq_r\ uq_i\ uq_j\ uq_k]^T,\ \ u \in \mathbb{R}, \tag{38}$$

which also coincides with the scalar product in $\mathbb{R}^4$.

However, the **quaternion product** is quite more different than what we do with vectors in $\mathbb{R}^4$. We use here the properties of the base $\{\mathbf{1}, \mathbf{i}, \mathbf{j}, \mathbf{k}\}$ defined above, and the distributive law:

$$\mathbf{q} = \mathbf{q}^a \cdot \mathbf{q}^b = \begin{bmatrix} q_r^a q_r^b - q_i^a q_i^b - q_j^a q_j^b - q_k^a q_k^b \\ q_r^a q_i^b + q_i^a q_r^b + q_j^a q_k^b - q_k^a q_j^b \\ q_r^a q_j^b - q_i^a q_k^b + q_j^a q_r^b + q_k^a q_i^b \\ q_r^a q_k^b + q_i^a q_j^b - q_j^a q_i^b + q_k^a q_r^b \end{bmatrix}. \tag{39}$$

Finally, the **conjugate** of a quaternion is denoted as $\mathbf{q}^*$ and is computed as:

$$\mathbf{q}^* = [q_r\ -q_i\ -q_j\ -q_k]^T \tag{40}$$

Definition **Unit quaternions** are those with unit norm, $|\mathbf{q}| = 1$. They are a way to represent rotations in 3D, usually not so intuitive, but with major practical benefits, specially in computer implementations. Let be a quaternion $\mathbf{q}_B^A$ a quaternion representing the rotation of frame $\mathscr{F}^B$ with respect to frame $\mathscr{F}^A$. Then, given a vector $\mathbf{v}^B$ expressed in terms of $\mathscr{F}^B$, its components in terms of $\mathscr{F}^A$ can be computed with the following equation:

$$\begin{bmatrix} 0 \\ v_x^A \\ v_y^A \\ v_z^A \end{bmatrix} = \mathbf{q}_B^A \cdot \begin{bmatrix} 0 \\ v_x^B \\ v_y^B \\ v_z^B \end{bmatrix} \cdot (\mathbf{q}_B^A)^* \tag{41}$$

where the two product involved in the right side of the equation are quaternion products.

Interpretation A good way to interpret 3D rotations represented by unit quaternions is to revisit how Complex numbers can represent 2D rotations. Let $\mathbf{c} = a + bi \in \mathbb{C}$ be a complex number with unit norm, $|\mathbf{c}| = 1$. In that case, $\mathbf{c}$ is representing a point in the unit circle lying on the 2D plane, so it is also representing an angle α (FigureXX). This complex number can be written as:

$$\mathbf{c} = \cos\alpha + \sin\alpha i \tag{42}$$

Quaternions are an extension of this idea but exported to 3D, so they require three imaginary parts to fully represent a rotation, instead of a single imaginary part required in 2D:

$$\mathbf{q} = \cos\frac{\alpha}{2} + u_x \sin\frac{\alpha}{2}\mathbf{i} + u_y \sin\frac{\alpha}{2}\mathbf{j} + u_z \sin\frac{\alpha}{2}\mathbf{k} \tag{43}$$

where α is the angle over the rotation axis indicated by the vector $\mathbf{u} = [u_x\ u_y\ u_z]^T$. Therefore, if we rotate a vector over the Euclidean X, Y or Z axes, we get a complex representations over each $x = 0, y = 0$ or $z = 0$ planes respectively.

Chain of Rotations Unit quaternions behave very similarly to rotation matrix when chaining rotations. Lets involve three frames $\mathscr{F}^A$, $\mathscr{F}^B$ and $\mathscr{F}^C$. $\mathbf{q}_B^A$ represents the rotation of frame B with respect to frame A, and $\mathbf{q}_C^B$ represents the one of frame C with respect to B. Then, $\mathbf{q}_C^A$, which is the rotation of frame C with respect to A is:

$$\mathbf{q}_C^A = \mathbf{q}_B^A \cdot \mathbf{q}_C^B \tag{44}$$

The operation required to compute $\mathbf{q}_C^A$ performs 16 multiplications and 12 additions, so it is more computationally efficient than matrix chaining. This can be of relevance in real-time algorithms requiring massive computtaion of chains of rotations.

Conversion to Rotation Matrix Sometimes it can be useful to convert a quaternion to a rotation matrix. The formula below provides such conversion:

$$\mathbf{R}(\mathbf{q}) = 2\begin{bmatrix} \frac{1}{2} - q_2^2 - q_3^2 & q_1q_2 - q_0q_3 & q_0q_2 + q_1q_3 \\ q_0q_3 + q_1q_2 & \frac{1}{2} - q_1^2 - q_3^2 & q_2q_3 - q_0q_1 \\ q_1q_3 - q_0q_2 & q_0q_1 + q_2q_3 & \frac{1}{2} - q_1^2 - q_2^2 \end{bmatrix}, \tag{45}$$

3.1.3 Axis Angle

//TODO

3.1.4 Euler Angles

Definition Euler angles approach is an intuitive way to represent rotations, by expressing the three rotation angles applied to different orthogonal axis, in a given order. One of the most used conventions is the $Z-Y-X$ order, leading to three angles:

- Yaw, θ, is the first angle, a rotation around the Z axis of the reference frame, $\theta \in (-\pi, \pi]$.
- Pitch, ϕ, is the second angle, a rotation around the current (once rotated) Y axis, $\phi \in (-\pi/2, \pi/2]$.
- Roll, ψ, is the third angle, a rotation around the current (twice rotated) X axis, $\psi \in (-\pi, \pi]$.

Figure 2 shows the order of the rotations of each euler angle to get the final orientation.

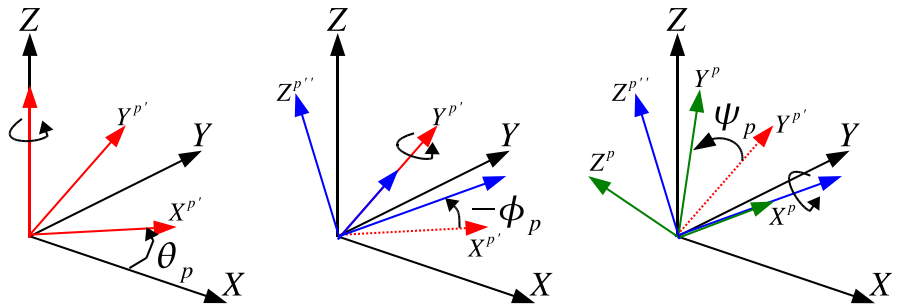

Figure 2: Definition of the three euler angles yaw (θ), pitch (ϕ) and roll (ψ). Reference frame drawn in black. Intermediate frames drawn in red and blue. Final orientation frame shown in green.

Conversion to Rotation Matrix Since rotations are performed always around the current axis instead of around the fixed world axis, the whole rotation matrix is computed as:

$$R(\theta,\phi,\psi) = R_{\theta,\phi,\psi} = R_z(\theta)R_y(\phi)R_x(\psi) \tag{46}$$

where,

$$\begin{aligned} R_z(\theta) &= \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}; \\ R_y(\phi) &= \begin{pmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{pmatrix}; \\ R_x(\psi) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{pmatrix}; \end{aligned} \tag{47}$$

3.2 Adding translation

Beyond rotations, a transformation may involve also a translation, which is the displacement between two frames, without taking into account their orientation. If two frames are translated, there is simply a vector to add between them:

$$\mathbf{v}^A = \mathbf{v}^B + \mathbf{p}_B^A \tag{48}$$

where $\mathbf{p}_B^A$ is the vector expressing central point of frame B in terms of frame A.

When frame B is both rotated and translated with respect to frame A, we apply first the rotation, and then the translation (see Figure 3):

$$\mathbf{v}^A = \mathbf{R}_B^A\mathbf{v}^B + \mathbf{p}_B^A \tag{49}$$

3.3 Homogeneous matrix

Homogeneous matrix is a compact way to represent both rotation and translation in a single matrix, by adding an extra dimensionality to the rotation matrix. The homogeneous matrix representing the frame B in terms of the frame A (both rotation and translation) is defined as:

$$\mathbf{T}_B^A = \begin{bmatrix} \mathbf{R}_B^A & \mathbf{p}_B^A \\ 0 & 1 \end{bmatrix} \tag{50}$$

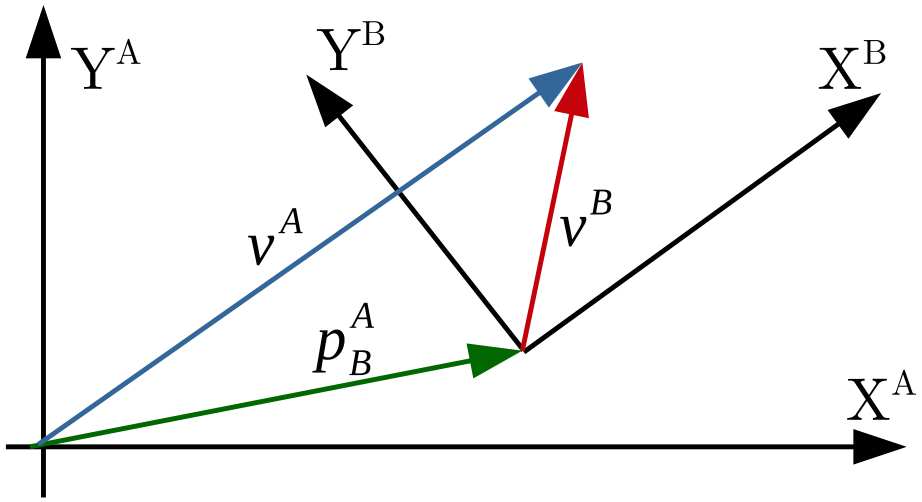

Figure 3: A frame B, rotated and translated with respect to frame A.

Then, as it was described by pure rotation case in subsection 3.1.1, it fulfills that:

$$\begin{bmatrix} \mathbf{v}^A \\ 1 \end{bmatrix} = \mathbf{T}_B^A \begin{bmatrix} \mathbf{v}^B \\ 1 \end{bmatrix}; \quad \begin{bmatrix} \mathbf{v}^B \\ 1 \end{bmatrix} = (\mathbf{T}_B^A)^{-1} \begin{bmatrix} \mathbf{v}^A \\ 1 \end{bmatrix}. \tag{51}$$

Homogeneous matrixes also allow chaining as rotations do:

$$\begin{bmatrix} \mathbf{v}^A \\ 1 \end{bmatrix} = \mathbf{T}_B^A \mathbf{T}_C^B \begin{bmatrix} \mathbf{v}^C \\ 1 \end{bmatrix} = \mathbf{T}_C^A \begin{bmatrix} \mathbf{v}^C \\ 1 \end{bmatrix}. \tag{52}$$

Example 3. Vehicle frames. As Figure 4 draws, imagine we have a vehicle well localized at point $p^O = \begin{bmatrix} p_x^O & p_y^O \end{bmatrix}^T$ and oriented an angle θ with respect to the origin of the trajectory. This vehicle has a sensor mounted at the known position $m^B = [m_x^B \; m_y^B]$ and oriented an angle β with respect to the base frame of the vehicle. Finally this sensor has detected a point of interest at $q^S = [q_x^S \; q_y^S]$. Which are the coordinates of the point of interest with respect to the base frame, and with respect to the origin of the trajectory ?

The rotation matrixes involved are:

$$\mathbf{R}_B^O = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}; \quad \mathbf{R}_S^B = \begin{bmatrix} cos\beta & -sin\beta \\ sin\beta & cos\beta \end{bmatrix}; \tag{53}$$

and the homogeneous matrixes representing the transformations from the origin to the base, and from the base to the sensor, are respectively:

$$\mathbf{T}_B^O = \begin{bmatrix} \mathbf{R}_B^O & p^0 \\ \mathbf{0} & 1 \end{bmatrix}; \quad \mathbf{T}_S^B = \begin{bmatrix} \mathbf{R}_S^B & m^B \\ \mathbf{0} & 1 \end{bmatrix}; \tag{54}$$

So the point q with respect to the vehicle base frame and with respect to the origin of the trajectory is:

$$\begin{bmatrix} \mathbf{q}^B \\ 1 \end{bmatrix} = \mathbf{T}_S^B \begin{bmatrix} \mathbf{q}_S \\ 1 \end{bmatrix}; \quad \begin{bmatrix} \mathbf{q}^O \\ 1 \end{bmatrix} = \mathbf{T}_B^O \mathbf{T}_S^B \begin{bmatrix} \mathbf{q}_S \\ 1 \end{bmatrix}; \tag{55}$$

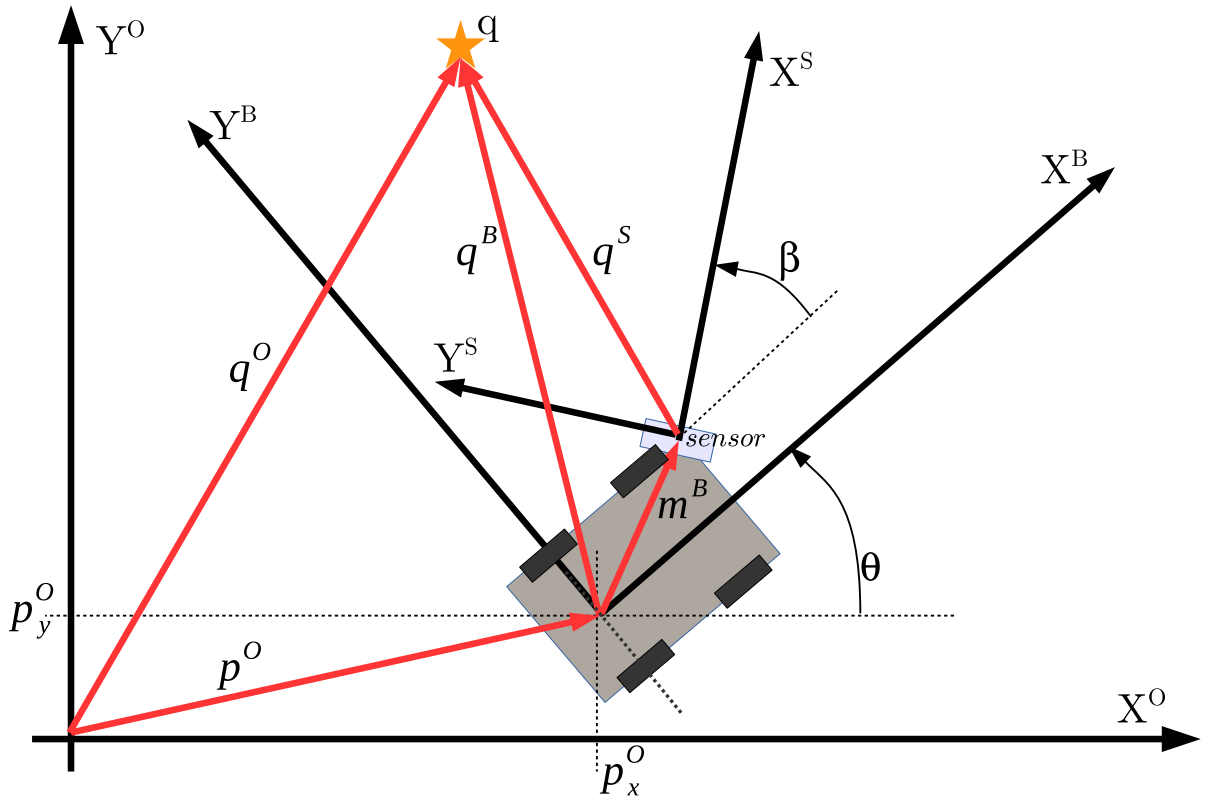

Figure 4: Trajectory, vehicle and sensor frames.

3.4 Choosing the right parameterization in 3D

- Nature and application of the vehicle/tool: ship, car, end-effector, camera, ...
- Differentation
- Dimensionality (Euler and quaternions only for 3D)
- ...

//TODO

4 Matrix decompositions

Matrix decompositions are a set of methods that allow to express a single matrix as a product of several other matrix that may have special properties, such as they can be, for instance diagonal, triangular or rotation matrixes, so we can them apply properties, usually to speed up computation, or to better interpret main directions/components of a given matrix.

4.1 Eigen Decomposition

//TODO

4.2 Singular Value Decomposition (SVD)

Definition Given a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$, the SVD decomposition is:

$$\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^T \tag{56}$$

where $\mathbf{U} \in \mathbb{R}^{m \times m}$ is a squared orthogonal matrix , $\mathbf{D} \in \mathbb{R}^{m \times n}$ is a rectangular diagonal matrix with non-negative entries in the diagonal and $\mathbf{V} \in \mathbb{R}^{n \times n}$ is also a squared orthogonal matrix. Diagonal entries of $\mathbf{D}$ are known as **singular values**, while the m columns of $\mathbf{U}$, as well as the n columns of $\mathbf{V}$ are called left-singular and right-singular vectors respectively.

Interpretation In the common case of $\mathbf{M} \in \mathbb{R}^{m \times m}$, with $|\mathbf{M}| > 0$, SVD can be interpreted as a composition of three geometrical transforms: a rotation expressed by $\mathbf{V}^T$, a scaling by $\mathbf{D}$ and last rotation by $\mathbf{U}$. Summarizing, $\mathbf{Ma}$ equals to:

$$\begin{aligned} \mathbf{V}^T\mathbf{a} &\rightarrow rotation \\ \mathbf{D}(\mathbf{V}^T\mathbf{a}) &\rightarrow scaling \\ \mathbf{U}(\mathbf{D}(\mathbf{V}^T\mathbf{a})) &\rightarrow rotation \end{aligned} \tag{57}$$

Singular values (entries at diagonal of $\mathbf{D}$) are the semiaxes of an ellipsoid in $\mathbb{R}^m$. Therefore, this decomposition indicates which are the orthogonal directions spanned by a matrix.

Example 4. SVD in 2D //TODO: Eigen and Scilab code Given the following matrix:

$$\mathbf{M} = \begin{bmatrix} 2 & 0.5 \\ 0.3 & 3 \end{bmatrix} \tag{58}$$

which is squared and with $|\mathbf{M}| > 0$. Computing its SVD with a computer library leads to the following result:

$$\mathbf{U} = \begin{bmatrix} 0.349802 & 0.936824 \\ 0.936824 & -0.349802 \end{bmatrix};\ \mathbf{D} = \begin{bmatrix} 3.142312 & 0. \\ 0. & 1.861687 \end{bmatrix};\ \mathbf{V} = \begin{bmatrix} 0.312080 & 0.950056 \\ 0.950056 & -0.312080 \end{bmatrix}; \tag{59}$$

We can check that $\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^T$

Relation with Eigen Decomposition The following relations hold between Eigen values and Singular values:

- The left-singular vectors of $\mathbf{M}$ are Eigenvectors of $\mathbf{M}\mathbf{M}^T$.
- The right-singular vectors of $\mathbf{M}$ are Eigenvectors of $\mathbf{M}^T\mathbf{M}$.
- The non-zero singular values of $\mathbf{M}$ (diagonal entries of $\mathbf{D}$), are the square root of the non-zero Eigenvalues of both $\mathbf{M}\mathbf{M}^T$ and $\mathbf{M}^T\mathbf{M}$.

4.3 Cholesky

Definition Cholesky decomposition allows to express a hermitian and positive-definite matrix $\mathbf{A}$ as:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^* \tag{60}$$

where $\mathbf{L}$ is a lower triangular matrix, with entries $l_{ij} \in \mathbb{R}^+$. Matrix $\mathbf{A}$ has a unique Cholesky decomposition.

Use in System Solving Given a linear system with vector $\mathbf{x}$ as the unknown:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{61}$$

It can be solved by executing the following steps:

- $\mathbf{L} \leftarrow Cholesky(\mathbf{A})$
- Solve $\mathbf{L}\mathbf{y} = \mathbf{b}$ by forward substitution.
- Solve $\mathbf{L}^*\mathbf{x} = \mathbf{y}$ by backward substitution.

4.4 QR

//TODO

5 Differentation and Linearization

Differentation is the main tool to analyse how sensitive is a function with respect to each of the input variables. In some cases it might be interesting to linearize functions to simplify computations in order to speed up processing loops.

5.1 Differentation

Let $f(\mathbf{x})$ be a scalar function, where $\mathbf{x} \in \mathbb{R}^n$. A **partial derivative** of $f()$ with respect to the $i-th$ component x_i is defined as:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \tag{62}$$

which is the magnitude of the slope of the function f alongside dimension x_i. This value can be interpreted as how much senisitve is the function f to changes in the $i-th$ component. Such measure of sensitivity is of major interest, for instance, to evaluate uncertainty effects. For the same function f, the **gradient** is defined as the following row vector:

$$\nabla_{f((x))} = \nabla_f = \left[\frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_j} \cdots \frac{\partial f}{\partial x_n}\right] \tag{63}$$

The gradient vector indicates the direction and magnitude of the slope of the function f at the point where it is evaluated, so it is also concept of major interest.

In the case where $f(\mathbf{x})$ is a vector function $\mathbb{R}^n \to \mathbb{R}^m$, it is also defined the **Jacobian** matrix as:

$$\mathbf{J}_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_j} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_2}{\partial x_j} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & \vdots & & \vdots \\ \frac{\partial f_i}{\partial x_1} & \cdots & \frac{\partial f_i}{\partial x_j} & \cdots & \frac{\partial f_i}{\partial x_n} \\ \vdots & & \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_j} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla_{f_1} \\ \nabla_{f_2} \\ \vdots \\ \nabla_{f_i} \\ \vdots \\ \nabla_{f_m} \end{bmatrix} \tag{64}$$

Finally, the **Hessian** matrix, defined for a function $f : \mathbb{R}^n \to \mathbb{R}$, is that of second order derivatives, defined as:

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial f}{\partial x_1 \partial x_j} & \cdots & \frac{\partial f}{\partial x_1 \partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_2}{\partial x_j} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & \vdots & & \vdots \\ \frac{\partial f_i}{\partial x_1} & \cdots & \frac{\partial f_i}{\partial x_j} & \cdots & \frac{\partial f_i}{\partial x_n} \\ \vdots & & \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_j} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{65}$$

Example 5. Jacobian of vehicle frames. The example illustrated with figure 4 computed the rigid transformation of a point $\mathbf{q}^S$ expressed with respect to a sensor frame, to the same point with respect to the frame at the origin of the trajectory, $\mathbf{q}^O$. The later can be seen as a function:

$$\mathbf{q}^O : \mathbb{R}^8 \to \mathbb{R}^2 \tag{66}$$

since it depends on eight variables: $q_x^S, q_y^S, m_x^B, m_y^B, \beta, p_x^O, p_y^O, \theta$ in the following way (from equation 55):

$$\begin{bmatrix} \mathbf{q}^O \\ 1 \end{bmatrix} = \mathbf{T}_B^O \mathbf{T}_S^B \begin{bmatrix} \mathbf{q}_S \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta+\beta) & -\sin(\theta+\beta) & m_x^B\cos\theta - m_y^B\sin\theta + p_x^O \\ \sin(\theta+\beta) & \cos(\theta+\beta) & m_y^B\sin\theta + m_y^B\cos\theta + p_y^O \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{q}_S \\ 1 \end{bmatrix} \quad (67)$$

$$\mathbf{q}^O = \begin{bmatrix} q_x^S\cos(\theta+\beta) - q_y^S\sin(\theta+\beta) + m_x^B\cos\theta - m_y^B\sin\theta + p_x^O \\ q_x^S\sin(\theta+\beta) + q_y^S\cos(\theta+\beta) + m_x^B\sin\theta + m_y^B\cos\theta + p_y^O \\ 1 \end{bmatrix} \quad (68)$$

So the full Jacobian of $\mathbf{q}^O$, $\mathbf{J}_{\mathbf{q}^O}$, is the following 2×8 matrix:

$$\mathbf{J}_{\mathbf{q}^O} = \begin{bmatrix} c(\theta+\beta) & -s(\theta+\beta) & c\theta & -s\theta & -q_x^S s(\theta+\beta) - q_y^S c(\theta+\beta) & 1 & 0 & -q_x^S s(\theta+\beta) - q_y^S c(\theta+\beta) - m_x^B s\theta - m_y^B c\theta \\ s(\theta+\beta) & c(\theta+\beta) & s\theta & -c\theta & q_x^S c(\theta+\beta) - q_y^S s(\theta+\beta) & 0 & 1 & q_x^S c(\theta+\beta) - q_y^S s(\theta+\beta) + m_x^B c\theta - m_y^B s\theta \end{bmatrix} \quad (69)$$

where $c\alpha = \cos(\alpha)$ and $s\alpha = \sin(\alpha)$. This Jacobian shows how sensitive is the function $\mathbf{q}^O$ to each of the eight input variables. It can be easily seen, for instance, that q_x^O is not sensitive to p_y^O. And also that when $\theta = 0$, q_y^O is not sensitive with m_x^B. Jacobians will be very useful in error propagation analysis as it will be seen in section 6.

5.2 One-dimensional Taylor's Theorem

From [xx], the Taylor's theorem for a one-dimensional function can be expressed as follows: Let $k \geq 1$ be an integer and let the function $f : \mathbb{R} \to \mathbb{R}$ be k times differentiable at the point $a \in \mathbb{R}$. Then there exists a function $h_k : \mathbb{R} \to \mathbb{R}$ such that:

$$f(x) = f(a) + \frac{\partial f}{\partial x}(a)(x-a) + \frac{1}{2!}\frac{\partial^2 f}{\partial x^2}(a)(x-a)^2 + \cdots + \frac{1}{k!}\frac{\partial^k f}{\partial x^k}(a)(x-a)^k + h_k(x)(x-a)^k. \quad (70)$$

and $\lim_{x\to\infty} h_k(x) = 0$.

Basically, the Taylor's theorem is saying that in the point $x = a$, a function $f(x)$ can be approximated by computing the successive derivatives, from order 0 (which is the constant $f(a)$) up to order k. Sometimes it is of great interest to have an approximation of a function around a point, so we can compute it fastly, or manipulate it easily. On particular case is when $k = 1$, which is called a *linearization* of the function, since the approximation takes into account just the first derivative $(k = 1)$, so we have:

$$f(x) \approx f(a) + \frac{\partial f}{\partial x}(a)(x-a) \quad (71)$$

Reacall that the approximation is only valid around the point $x = a$.

5.3 Multi-dimensional Taylor's Theorem (1st order)

Following the same idea, the Taylor's theorem is formulated for multi-variate functions such as:

$$f : \mathbb{R}^n \to \mathbb{R}^m \quad (72)$$

then the Taylor's Theorem up to the first order derivatives (linearization of f around the point $\mathbf{a}$) is:

$$f(\mathbf{x}) \approx f(\mathbf{a}) + \mathbf{J}_f(\mathbf{a})(\mathbf{x}-\mathbf{a}) \quad (73)$$

where $\mathbf{J}_f(\mathbf{a})$ is the Jacobian of function $\mathbf{f}$ evaluated at point $\mathbf{a}$.

5.4 Linearization error

6 Random Variables

A **random variable**, $\tilde{x}$, is a mathematical representation of a value and its associated uncertainty. Usually, we are interested not just in the behaviour of a certain value of a variable or parameter, but also in how it behaves when it is uncertain. A *sample* of such random variable can take several values according some **distribution law**. So we are specially interested in how this distribution law *reshapes* when passing through different physical processes and/or computation steps. The **probability density function**, also known as PDF, or just *density*, is a function

$$F_{\tilde{x}}(x) : \mathbb{R} \to \mathbb{R} \tag{74}$$

that expresses which values in the domain of the variable x are more likely, according to the random variable $\tilde{x}$. However, since this function is a density, the actual value expressing a probability is the integral between an interval:

$$p(a < \tilde{x} < b) = \int_a^b F_{\tilde{x}}(x)dx \tag{75}$$

A **random vector**, $\tilde{\mathbf{x}}$, is an array of random variables, stacked forming a column vector:

$$\tilde{\mathbf{x}} = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \vdots \\ \tilde{x}_n \end{bmatrix}. \tag{76}$$

In such case, the probability density function is defined as:

$$F_{\tilde{\mathbf{x}}}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R} \tag{77}$$

and the probability in a given interval, I, is computed as:

$$p(\tilde{\mathbf{x}} \in I) = p(a_n < \tilde{x}_1 < b_n, \ldots, a_n < \tilde{x}_n < b_n) = \int_{a_1}^{b_1} \cdots \int_{a_n}^{b_n} F_{\tilde{\mathbf{x}}}(\mathbf{x})dx_n \ldots dx_1 \tag{78}$$

6.1 Moments

A way to parameterize a distribution law is through its *moments*, a set of parameters that can fully describe the distribution in some cases, or just approximate it in others. The most known moments are the mean, μ, which is the fisrt order moment, and the variance, σ^2, which is the second order moment.

In the unidimensional case, from a set of p samples $\{x_1 \ldots x_p\}$, $x_i \in \mathbb{R}$, the mean and variance can be estimated as:

$$\mu_x = \frac{1}{p}\sum_{k=1}^{p} x_k; \quad \sigma_x^2 = \frac{1}{p-1}\sum_{k=1}^{p}(x_k - \mu_x)^2; \tag{79}$$

In the multivariate case the mean becomes a vector, $\boldsymbol{\mu}_x \in \mathbb{R}^n$. Given p samples, it can be estimated as:

$$\boldsymbol{\mu}_x = \frac{1}{p}\sum_{k=1}^{p} \mathbf{x}_k; \tag{80}$$

In that multivariate case, the second order moment can be *cross*computed, so a *covariance* matrix is used to describe the second order stathistics, which has the following squared form:

$$\mathbf{C}_x = \begin{bmatrix} c_{11} & \ldots & \ldots & c_{1j} & \ldots & c_{1n} \\ \vdots & & & \vdots & & \vdots \\ c_{i1} & \ldots & \ldots & c_{ij} & \ldots & c_{in} \\ \vdots & & & \vdots & & \vdots \\ c_{n1} & \ldots & \ldots & c_{nj} & \ldots & c_{nn} \end{bmatrix}; \tag{81}$$

From a set of p samples, the element c_{ij} of the covariance matrix is estimated as follows:

$$c_{ij} = \frac{1}{p-1} (\mathbf{s}_{x_i} - \mu_{x_i})^T \cdot (\mathbf{s}_{x_j} - \mu_{x_j}) \tag{82}$$

where $\mathbf{s}_{x_i}$ is a vector stacking all p samples of component i of the random variable $\tilde{\mathbf{x}}$, and a subtracting a scalar from a vector means that we substract the scalar at all components of the vector.

Interpretation of Covariance Matrix Recalling the interpretation of the scalar product as a measure of similarity between two vectors (subsection 2.2), the element c_{ij} of a covariance matrix is encoding the alignment between the components i and j of the random variable $\tilde{\mathbf{x}}$ in the set of p samples, once the mean is subtracted. Two components that suffer from the similar variations in the sample set, will appear as two vectors $\mathbf{s}_{x_i}$ and $\mathbf{s}_{x_j}$ nearly aligned, so its covariance c_{ij} will be close to $\frac{1}{p-1}$. In contrast, two components of the random variable $\tilde{\mathbf{x}}$ that change without any relation, lead to two vectors $\mathbf{s}_{x_i}$ and $\mathbf{s}_{x_j}$ unaligned, so the scalar product, and thus the c_{ij}, will approach to 0.

So, the covariance matrix builds a base of n vectors (not necessarily orthogonal!) in the space of the random variable $\tilde{\mathbf{x}}$ with directions according how individual components behave similarly or not in the dataset of p samples. SVD decomposition can be applied to find out the main orthogonal directions of this base.

Example 6. Compute a 2D covariance matrix from data [Scilab]. In this example we'll generate two sample sets S_1 and S_2, each one composed by 100 samples in $\mathbb{R}^2$, so 100 points (x,y). S_1 is built forcing a strong linear relation between first and second component, while S_2 is just computed by drawing random values for both components. The code below computes the covariance matrix for both sets.

```
//user entries
mm = 0.3;
bb = 0;
noise_stdev = 0.3; //sqrt(noise_variance)

//create set S1
xx1 = [0:0.1:10]';
[nn cols] = size(xx1); //get set size
yy1 = mm*xx1 + bb + noise_stdev*rand(nn,1,"normal");

//create set S2
xx2 = noise_stdev*rand(nn,1,"normal");
yy2 = noise_stdev*rand(nn,1,"normal");

//compute means, each component, each set:
mx1 = sum(xx1)/nn;
my1 = sum(yy1)/nn;
mx2 = sum(xx2)/nn;
my2 = sum(yy2)/nn;

//compute covariance matrix explicitly
cxx1 = (xx1-mx1)'*(xx1-mx1)/(nn-1);
cyy1 = (yy1-my1)'*(yy1-my1)/(nn-1);
cxy1 = (xx1-mx1)'*(yy1-my1)/(nn-1);
cxx2 = (xx2-mx2)'*(xx2-mx2)/(nn-1);
cyy2 = (yy2-my2)'*(yy2-my2)/(nn-1);
cxy2 = (xx2-mx2)'*(yy2-my2)/(nn-1);

//compute covariance matrix with scilab call
C1 = cov(xx1,yy1);
C2 = cov(xx2,yy2);
```

The resulting covariance matrices are:

$$\mathbf{C}_1 = \begin{bmatrix} 8.585 & 2.6312156 \\ 2.6312156 & 0.8963398 \end{bmatrix}; \ \mathbf{C}_2 = \begin{bmatrix} 0.0930911 & -0.0006647 \\ -0.0006647 & 0.0884355 \end{bmatrix}; \tag{83}$$

The resulting matrices show how the set S_1 presents a much higher relation between x and y components than S_2, due to the linear relation imposed when building the sets. Moreover, the fact that the values in C_1 are much larger is caused because S_1 stretches in the XY plane occupying a larger region than S_2.

Example 7. Ellipses from a 2D covariance matrix. This example shows a Scilab function to draw a 2D ellipses from a covariance matrix, centered at a given (x,y) point. Numerical errors can lead to non-positive-definite covariance matrixes after some computational steps. Therefore, the first calls in the function are related to ensure that the matrix is strictly positive-definite, a necessary condition to assure eigenvalues are real (TBC!!). Positive-definite is ensured through a Cholesky decomposition. Thereafter, eigenvalues are computed, which directly provide the length of a major and minor axes of the ellipses. Finally, drawing is performed point by point over the ellipses, with steps of 0.1 radian.

```
//covariance matrix Cmat, and centered at point mu
function[] = draw_ellispes_from_cov(mu, Cmat, axes_h)

    //ensure positive-definite matrix
    Cchol = chol(Cmat);
    CC = Cchol'*Cchol;

    //compute eigenvalues
    [RR,diagCC] = spec(CC);
    eval1 = diagCC(1,1);
    eval2 = diagCC(2,2);

    //sort evals by value. Set major and minor axes.
    if (eval1>eval2) then
        axis = [eval1;eval2;atan(RR(2,1))];
    else
        axis = [eval2;eval1;atan(RR(2,2))];
    end

    //start drawing (compute all points)
    step = 0.1; //Set drawing step
    t = 0:step:%pi/2; //set drawing vector
    eX = axis(1)*cos(t); //ellispes points (X component, a quarter)
    eY = axis(2)*sin(t); //ellipses points (Y component, a quarter)
    nn = 4*size(eX,'*'); //num of total points of the ellipses
    eXY1 = [eX, -flipdim(eX,2), -eX, flipdim(eX,2); eY, flipdim(eY,2), -eY, -flipdim(eY,2)];
    eXY = rotate(eXY1, axis(3)) + [mu(1)*ones(1,nn);mu(2)*ones(1,nn)];
    sca(axes_h); //Set current axes
    xpoly(eXY(1,:), eXY(2,:));

endfunction
```

6.2 Uniform distribution

A random variable or vector is called uniform when the density is constant over a bounded interval, meaning that the variable is equally distributed within this interval. That implies also that the variable can take, with equal probability, values within this interval.

Uniform random variables can be described with just two parameters: x_{min}, x_{max}. The mean and the variance values of an unfiorm random variable are:

$$\mu_{\tilde{x}} = \frac{x_{min} + x_{max}}{2}; \;\; \sigma_{\tilde{x}}^2 = \frac{1}{12}(x_{max} - x_{min})^2 \tag{84}$$

6.3 Gaussian distribution

A random variable is called Gaussian or Normal, when its density is fully described with two parameters, $\mu_{\tilde{x}}$ (mean) and $\sigma_{\tilde{x}}^2$ (standard deviation), and the following expression:

$$F_{\tilde{x}}(x) = \frac{1}{2\pi} e^{-(\frac{x-\mu_{\tilde{x}}}{\sigma_{\tilde{x}}})^2} \tag{85}$$

Gaussian variables are also written as $\mathcal{N}(\mu, \sigma)$, which is a widely used notation stressing the dependency with just two parameters. Given $\tilde{x} = \mathcal{N}(\mu_x, \sigma_x)$ and $\tilde{y} = \mathcal{N}(\mu_y, \sigma_y)$, the following properties fulfill:

- $\tilde{z} = a\tilde{x} \;\rightarrow \tilde{z} = \mathcal{N}(a\mu_x, a\sigma_x)$
- $\tilde{z} = \tilde{x} + \tilde{y} \;\rightarrow \tilde{z} = \mathcal{N}(\mu_x + \mu_y, \sqrt{\sigma_x^2 + \sigma_y^2})$

The two properties listed above are of major importance since they imply that we know how Gaussian variables behave when they pass through linear systems.

6.4 Multivariate Gaussian distribution

In case of n-dimensional Gaussian variables, the mean is $\boldsymbol{\mu} \in \mathbb{R}^n$, and the covariance is represented as a squared matrix, $\mathbf{C} \in \mathbb{R}^{n \times n}$. The generalized formula is the following:

$$F_{\tilde{x}}(\mathbf{x}) = \frac{1}{\sqrt{|\mathbf{C}|}(2\pi)^n} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{86}$$

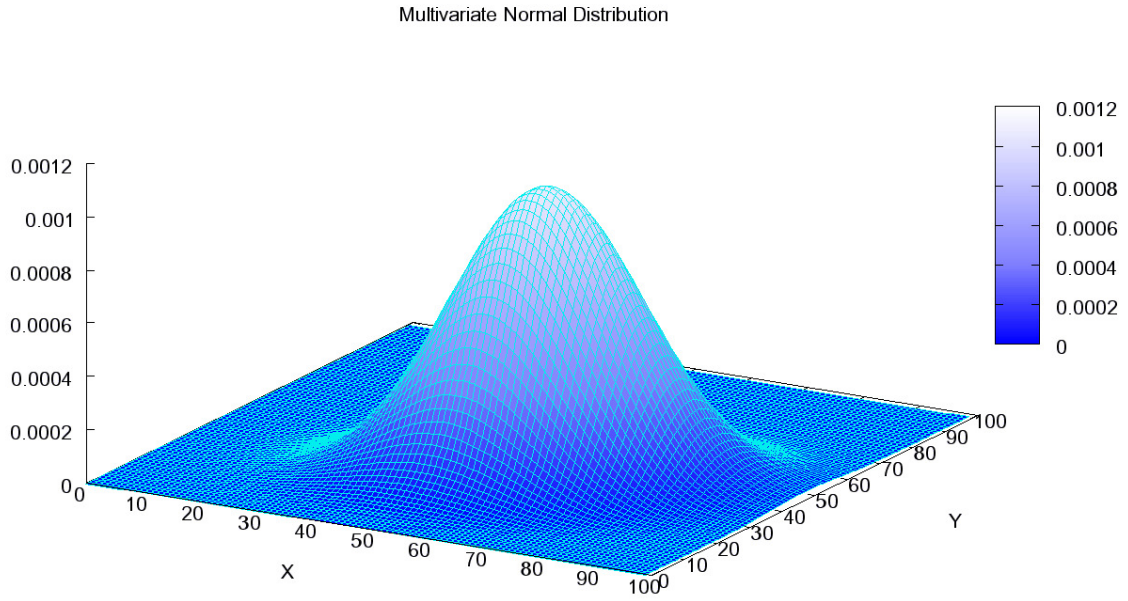

Figure 5: Bivariate Gaussian function

In the multivariate case, the properties around the sum and the linear operation also fulfill:

- $\tilde{\mathbf{z}} = \mathbf{A}\tilde{\mathbf{x}} \rightarrow \tilde{\mathbf{z}} = \mathcal{N}(\mathbf{A}\boldsymbol{\mu}_x, \mathbf{A}\mathbf{C}_x\mathbf{A}^T)$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$.
- $\tilde{\mathbf{z}} = \tilde{\mathbf{x}} + \tilde{\mathbf{y}} \rightarrow \tilde{\mathbf{z}} = \mathcal{N}(\boldsymbol{\mu}_x + \boldsymbol{\mu}_y, \mathbf{C}_x + \mathbf{C}_y)$

6.5 Gaussian Uncertainty Propagation

How uncertainty is propagated through different physical or computing processes is a major topic in robotics, specially in perception, but also in precise actuation. Specifically, we are interested in know how a density function reshapes due to some computation, view point change, ...

In previous subsections we've seen that Gaussian random variables and vectors have the great property that it is easy (not only conceptually but also computationally) to compute their parameters once they pass through linear operations (product by scalar and sum). We've also seen in section 5 that non-linear functions can be linearized, by assuming a linearization error. Therefore, linearization and Gaussian uncertainty propagation are two operations very common in many robotics algorithms. At intuitive level, linearization remains valid as long as uncertainty is enough *small* with respect to how the function changes (indicated by Jacobian).

Example 8. Vehicle frames revisited (with uncertainty) Let's come back to the example presented in subsection 3.3. However, now we will focus only in the system vehicle-sensor-point, but we will introduce two uncertainty sources: sensor measurement noise and sensor mounting point innacuracies (which can be also called extrinsics calibration uncertainty).

First of all, the point $\mathbf{q}$ is reported by a sensor in polar coordinates (q_r, q_α). The datasheet of that sensor indicates standard deviations for range measurement and azimuth σ_r and σ_α respectively. So the covariance matrix of a point q at the measurement space (r, α) is:

$$\mathbf{C}_{q^{r\alpha}} = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\alpha^2 \end{bmatrix} \tag{87}$$

Point $\mathbf{q}$ in the 2D-cartesian space, with respect to the sensor frame, is computed as:

$$\mathbf{q}^S = \begin{bmatrix} q_x^S \\ q_y^S \end{bmatrix} = \begin{bmatrix} q_r \cos(q_\alpha) \\ q_r \sin(q_\alpha) \end{bmatrix} \tag{88}$$

The equation above is not linear in the random variables, $[q_r, q_\alpha]^T$, so to propagate the uncertainty from the measurement space (r, α), to the 2D-cartesian one (x, y), we have to linearize this equation. From the section 5, we learned that multivariate functions require to compute the Jacobian in order to linearize the function around a point of interest, (q_r, q_α):

$$\mathbf{J}_{r\alpha}^{\mathbf{q}^S} = \begin{bmatrix} \cos(q_\alpha) & -q_r \sin(q_\alpha) \\ \sin(q_\alpha) & q_r \cos(q_\alpha) \end{bmatrix} \tag{89}$$

Once we know the (approximative) linear relation between $\mathbf{q}^S$ and $\mathbf{q}^{r\alpha}$, we can propagate the uncertainty from the polar (r, α) space to the 2D-cartesian (x, y) space, by using the properties of how Gaussian ranodm variables modify through linear systems, as discussed in subsection 6.4:

$$\mathbf{C}_{\mathbf{q}^S} = \mathbf{J}_{r\alpha}^{\mathbf{q}^S} \mathbf{C}_{\mathbf{q}^{r\alpha}} (\mathbf{J}_{r\alpha}^{\mathbf{q}^S})^T \tag{90}$$

where the Jacobian has to be evaluated at the point (q_r, q_α).

At this point we have propagated the the uncertainty of the point measurement in sensor frame from the polar space to the 2D-cartesian space. But we want to compute the uncertainty of the point $\mathbf{q}$ with respect to the vehicle frame. According to the example of vehicle frames, the expression of $\mathbf{q}^B$ is:

$$\begin{bmatrix} \mathbf{q}^B \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta & m_x^B \\ \sin\beta & \cos\beta & m_y^B \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_x^S \\ q_y^S \\ 1 \end{bmatrix} = \begin{bmatrix} q_x^S \cos\beta - q_y^S \sin\beta + m_x^B \\ q_x^S \sin\beta + q_y^S \cos\beta + m_y^B \\ 1 \end{bmatrix} \tag{91}$$

which depends on 5 uncertain variables: $(q_x^S, q_y^S, m_x^B, m_y^B, \beta)$. Again, the relation is not liinear with all the involved variables, so it is required to compute the Jacobian of the above expression, in order to propagate uncertainty from the involved variables to the final resulting point $\mathbf{q}^B$. This Jacobian, $\mathbf{J}_{q^S, m^B, \beta}$, a 2×5 matrix:

$$\mathbf{J}_{q^S, m^B, \beta} = \begin{bmatrix} \cos\beta & -\sin\beta & 1 & 0 & -q_x^S \sin\beta - q_y^S \cos\beta \\ \sin\beta & \cos\beta & 0 & 1 & q_x^S \cos\beta - q_y^S \sin\beta \end{bmatrix} \tag{92}$$

Covariance matrix related to (q_x^S, q_y^S) is $\mathbf{C}_{\mathbf{q}^S}$, and was computed previously. Uncertainty of the sensor mounting point is represented with the covariance matrix related to (m_x^B, m_y^B, β):

$$\mathbf{C}_{m^B, \beta} = \begin{bmatrix} \sigma_{m_x}^2 & 0 & 0 \\ 0 & \sigma_{m_y}^2 & 0 \\ 0 & 0 & \sigma_\beta^2 \end{bmatrix} \tag{93}$$

So the final covariance matrix of the point $\mathbf{q}$, with respect to the vehicle frame, $\mathbf{q}^B$, is:

$$\mathbf{C}_{\mathbf{q^B}} = \mathbf{J}_{q^S,m^B,\beta} \begin{bmatrix} \mathbf{C}_{\mathbf{q}^S} & [0]_{2\times 3} \\ [0]_{3\times 2} & \mathbf{C}_{m^B,\beta} \end{bmatrix} \mathbf{J}^T_{q^S,m^B,\beta} \tag{94}$$

The SciLab code below implements all this example and plots the ellispes related to $\mathbf{C}^{\mathbf{S}}_{\mathbf{q}}$ and $\mathbf{C}^{\mathbf{B}}_{\mathbf{q}}$.

```
 //clear
clear;

//include files (where draw_ellispes_from_cov() function is defined)
exec("/home/andreu/dev/uncertainty_propagation/ellipsesAxis.sci");

//Sensor point q detection in polar coordinates (r,a) (measurement space)
r_q = 8;
a_q = 23*%pi/180; //rad  (20.467)
qS = [r_q*cos(a_q); r_q*sin(a_q);1]; //point q in homogeneous coordinates wrt to the Sensor

//sensor noise in polar coordinates (measurement space)
sigma_range = 0.03; //meters
sigma_angle = 0.1*%pi/180; //rad
Cra_q = [sigma_range^2 0;0 sigma_angle^2]; //covariance matrix in measurement space
J_ra = [cos(a_q) -r_q*sin(a_q); sin(a_q) r_q*cos(a_q); 0 0]; //Jacobian: Linearization from measurement to homogeneous space

//Sensor mounting point with respect to the vehicle base
betaB_S = 35*%pi/180; //orientation angle of the sensor wrt the base
mB_S = [31;12]; //xy coordinates of the sensor wrt the base
RB_S = [cos(betaB_S) -sin(betaB_S); sin(betaB_S) cos(betaB_S)]; //rotation of the sensor wrt the base (R base2sensor)
TB_S = [RB_S mB_S;0 0 1]; //homogeneous transform of the sensor wrt the base (T base2sensor)

//sensor mounting point uncertainty (calibration error, or on-line sensor frame positioning error)
sigma_mx = 0.005; //meters
sigma_my = 0.005; //meters
sigma_beta = 0.1*%pi/180; //rad
C_mbeta = [sigma_mx^2 0 0; 0 sigma_my^2 0; 0 0 sigma_beta^2];
J_mbeta = [1 0 -qS(1)*sin(betaB_S)-qS(2)*cos(betaB_S); 0 1 qS(1)*cos(betaB_S)-qS(2)*sin(betaB_S); 0 0 0];

//------------------- PROPAGATE COVARIANCES --------------
CS_q = J_ra*Cra_q*J_ra'; disp(CS_q);
CB_q = TB_S*CS_q*TB_S' + J_mbeta*C_mbeta*J_mbeta'; disp(CB_q);

//-------------------- DRAW ELLIPSES ----------------------
figure('BackgroundColor',[1 1 1]);
drawaxis();
ph = gca(); // handle
ph.isoview = 'on';
ph.axes_visible = ["on","on","off"]
ph.grid = [1,1];
ph.auto_scale="on";
ph.auto_clear = 'off';

//CS_q
draw_ellispes_from_cov([0 0], CS_q(1:2,1:2),ph);
e=gce(); // get the current entity (the last created, the ellipses)
set(e,"foreground",1);
set(e,"thickness",3);
ph.auto_clear = 'off';

//CB_q
draw_ellispes_from_cov([0 0], CB_q(1:2,1:2),ph);
e=gce(); // get the current entity (the last created, the ellipses)
set(e,"foreground",2);
set(e,"thickness",3);
```

For the values of the code example above, the two plotted ellipses are shown in Figure 6. In the black ellispes related to the point uncertainty with respect to the sensor frame, $\mathbf{C}^{\mathbf{S}}_{\mathbf{q}}$, it can be seen how the shape of the uncertainty is larger in the range measurement component (r), rather than in the angular measurement (α). The blue ellispes shows the effect of a rotation provoked by the frame transformation ($\mathbf{T}^B_S$), but also the growing due to a new source of uncertainty due to errors on teh calibration mounting pose.

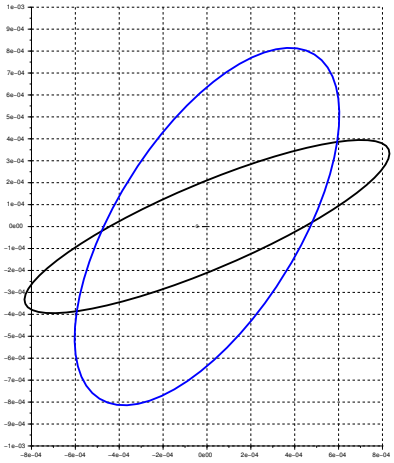

Figure 6: Ellipses related to covariance matrixes $\mathbf{C_q^S}$ (black) and $\mathbf{C_q^B}$ (blue).

Part II
Kinematics

7 Wheeled Vehicle Kinematics

This section is about how coordinate frames move according the geometry, velocities, rotational rates, positions and orientations of their mechanical elements. The section puts the focus on different configurations of wheeled vehicles. The goal of each subsection is to find out the relation between the actuators (usually wheel rotation rates and/or steering angles) and the kinematic state of the vehicle, which is usually a 2D twist (v_x, v_y, w_z), corresponding to forward velocity, lateral velocity and rotational rate respectively. To simplify the notation, this 2D twist will be written as (u, v, w) through this section.

7.1 Forward and inverse kinematics

Two relations are of interest between actuators and vehicle twist: *forward* and *inverse* kinematics.

Forward kinematics gets actuator values as inputs and computes the twist, while inverse kinematics computes the actuator values given a twist as input. The former is generally of interest in estimation and prediction problems, while the later is usually of interest in planning and control.

7.2 The Coriolis Law

Given two coordinate frames, one called *moving*($\mathscr{F}^M$) which is rotating at angular velocity $\mathbf{w}$ about the other frame, called *fixed*($\mathscr{F}^F$). Then the derivative of any vector $\mathbf{x}$ with respect to the time is related by:

$$\frac{d\mathbf{x}^F}{dt} = \frac{d\mathbf{x}^M}{dt} + \mathbf{w} \times \mathbf{x}^F \tag{95}$$

where $\mathbf{x}$ can be , for instance, position, velocity or force.

In the practical situation of a vehicle rotating about a fixed frame of reference, a given point $\mathbf{p}$ of that vehicle (fixed with respect to the vehicle frame) has a null time derivative in the vehicle frame ($\frac{d\mathbf{p}^M}{dt} = 0$), so equation 95 simplifies to the expression of the linear velocity of any point of the vehicle according to the rotation of the vehicle and the position of that point with respect to fixed frame.

$$\mathbf{v} = \frac{d\mathbf{p}^F}{dt} = \mathbf{w} \times \mathbf{p}^F \tag{96}$$

7.3 Instantaneous Center of Rotation (ICR)

From mechanics, we know that all planar motions of a rigid body can be described as a pure rotation about some point, called the instantaneous center of rotation (ICR). All points of the rigid body can describe its motion as a rotation about the same ICR.

For kinematics analysis, it is of special interest the contact point of the wheels with the floor, since these points are fixed in the vehicle frame and they also have the wheel restriction of no slippage.

7.4 Single Wheel

The most basic mechanical element of wheeled vehicles are their wheels, so let's take a look on how they work from the kinematics point of view.

A wheel is actuated with a motor providing a rotation rate to its axis, Ω. This rotation rate, will cause a linear velocity of the wheel center as:

$$u = \Omega r \tag{97}$$

where r is the wheel radius.

Despite forward velocity, a single wheel has $v = w = 0$, since it is not actuated to cause motion other than forward (1D case). It is a straightforward example of how the geometry of a body shapes the relation between the actuation variable (rotational rate, Ω), and the derived state of the 'vehicle' (linear velocity, u)

7.5 Bicycle

The bicycle case is the platform composed by two wheels mounted one in front of the other, separated by a distance L. The rear wheel drives the vehicle, while the front one steers it. Figure 7 shows this configuration. Actuator inputs are rear wheel rotation rate, Ω, and front wheel steering angle, α. Con-

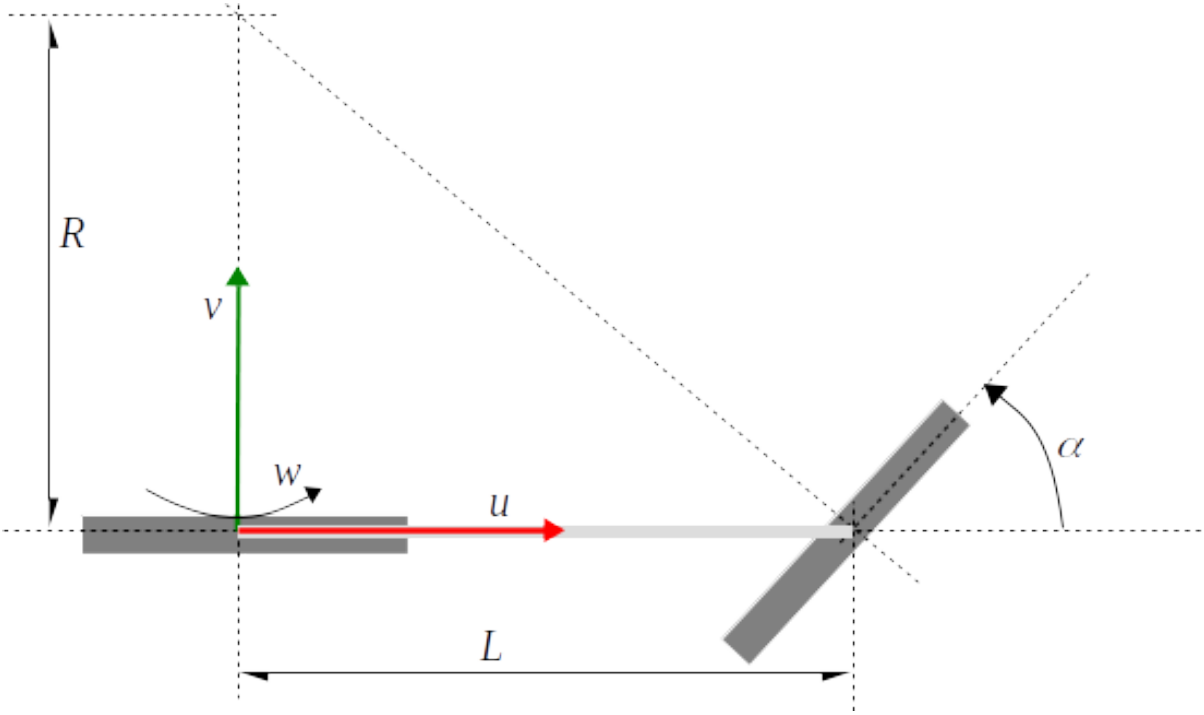

Figure 7: Bicycle configuration.

structive parameters are the distance between wheels ,L, and the rear wheel radius, r. Parameter R in the figure 7 is the curvature radius, which is an intermediate parameter of ineterst.

The linear velocities are:

$$u = \Omega r \tag{98}$$

$$v = 0 \tag{99}$$

and the vehicle should fulfill:

$$u = wR. \tag{100}$$

Given

$$\tan\alpha = \frac{L}{R} \tag{101}$$

and matching the two previous expressions for u:

$$u = \Omega r = wR = w\frac{L}{\tan\alpha} \tag{102}$$

leads to the expression for vehicle rotation rate:

$$w = \frac{\Omega r}{L} \tan \alpha \tag{103}$$

In this case the forward kinematics is not linear, since the w part of the twist involve a $\tan()$ relation with the input parameter α.

For several purposes it might be interesting to linearize this relation between input actuators, constructive parameters and output twist(see section 5 for linearization overview). Starting with the linearization with respect to the input actuators α and Ω, the Jacobian is:

$$\mathbf{J}_{\alpha\Omega} = \begin{bmatrix} 0 & r \\ 0 & 0 \\ \frac{\Omega r}{L\cos^2 \alpha} & \frac{r}{L}\tan\alpha \end{bmatrix}; \tag{104}$$

so forward kinematics relation can be approximated by the linearization as follows:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} \approx \mathbf{J}_{\alpha\Omega} \begin{bmatrix} \alpha \\ \Omega \end{bmatrix} \tag{105}$$

which can be useful, for instance, for Kalman filtering (see section 9).

If the interest is to analyze how sensitive is the kinematics to the constructive parameters, the goal is to compute the Jacobian matrix with respect to the parameters r and L, and perform an error propagation analysis. First the linearized relation is:

$$\mathbf{J}_{rL} = \begin{bmatrix} \Omega & 0 \\ 0 & 0 \\ \frac{\Omega}{L}\tan\alpha & -\frac{\Omega r}{L^2}\tan\alpha \end{bmatrix}; \tag{106}$$

Given an uncertainty in the r and L parameters, represented with a covariance matrix as:

$$\mathbf{C}_{rL} = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_L^2 \end{bmatrix}; \tag{107}$$

the propagation of such uncertainty to twist space is:

$$\mathbf{C}_{uvw} = \mathbf{J}_{rL}\mathbf{C}_{rL}\mathbf{J}_{rL}^T = \begin{bmatrix} \Omega & 0 \\ 0 & 0 \\ \frac{\Omega}{L}\tan\alpha & -\frac{\Omega r}{L^2}\tan\alpha \end{bmatrix} \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_L^2 \end{bmatrix} \begin{bmatrix} \Omega & 0 & \frac{\Omega}{L}\tan\alpha \\ r & 0 & -\frac{\Omega r}{L^2}\tan\alpha \end{bmatrix}; \tag{108}$$

$$\mathbf{C}_{uvw} = \begin{bmatrix} \sigma_r^2\Omega^2 & 0 & \sigma_r^2\frac{\Omega^2}{L}\tan\alpha \\ 0 & 0 & 0 \\ \sigma_r^2\frac{\Omega^2}{L}\tan\alpha & 0 & \frac{\Omega^2}{L^2}\tan^2\alpha(\sigma_r^2 + \frac{r^2}{L^2}\sigma_L^2) \end{bmatrix}; \tag{109}$$

Meaning that for a reasonable uncertainty values of $\sigma_r^2 = 10^{-6}\ m^2$ and $\sigma_L^2 = 10^{-6}\ m^2$ (1 mm of standard deviation in both cases r and L), and evaluating at $\Omega = 2\pi\ rad/s$, $\alpha = 45^o$, $r = 0.4\ m$ and $L = 1\ m$, the standard deviations in linear speed and rotational rate of the vehicle are:

$$\sigma_u \approx 6\ mm/s; \quad \sigma_w \approx 4\ mrad/s; \tag{110}$$

Inverse kinematics Inverse kinematics for the bicycle configuration is found from equations 98 and 103 as follows:

$$\Omega = \frac{u}{r} \tag{111}$$

$$\alpha = \arctan(L\frac{w}{u}), \quad \alpha \in [-\pi/2, \pi/2] \tag{112}$$

7.6 Tricycle

The tricycle has two configurations, depending if the drive actuator is at the front axis or at the back one. In the former configuration, the kinematics relation is exactly those previously analyzed for the monocycle case. However, in the later configuration, the driving actuator is attached at the rear axis, providing a forward velocity, while the front wheel just steers the vehicle. So, this configuration is like the bicycle case analyzed above, but some extra differential mechanism is required to drive each rear wheel at different speeds when turning, otherwise the wheels would slip.

7.7 Two wheels differential drive

Two wheels differential drive configuration is an essential and widely used architecture, due to its building simplicity and good manoeuvring properties, since it allows the platform to turn on the spot.

Figure 8 shows the two wheels differential drive configuration as well as the involved parameters and variables.

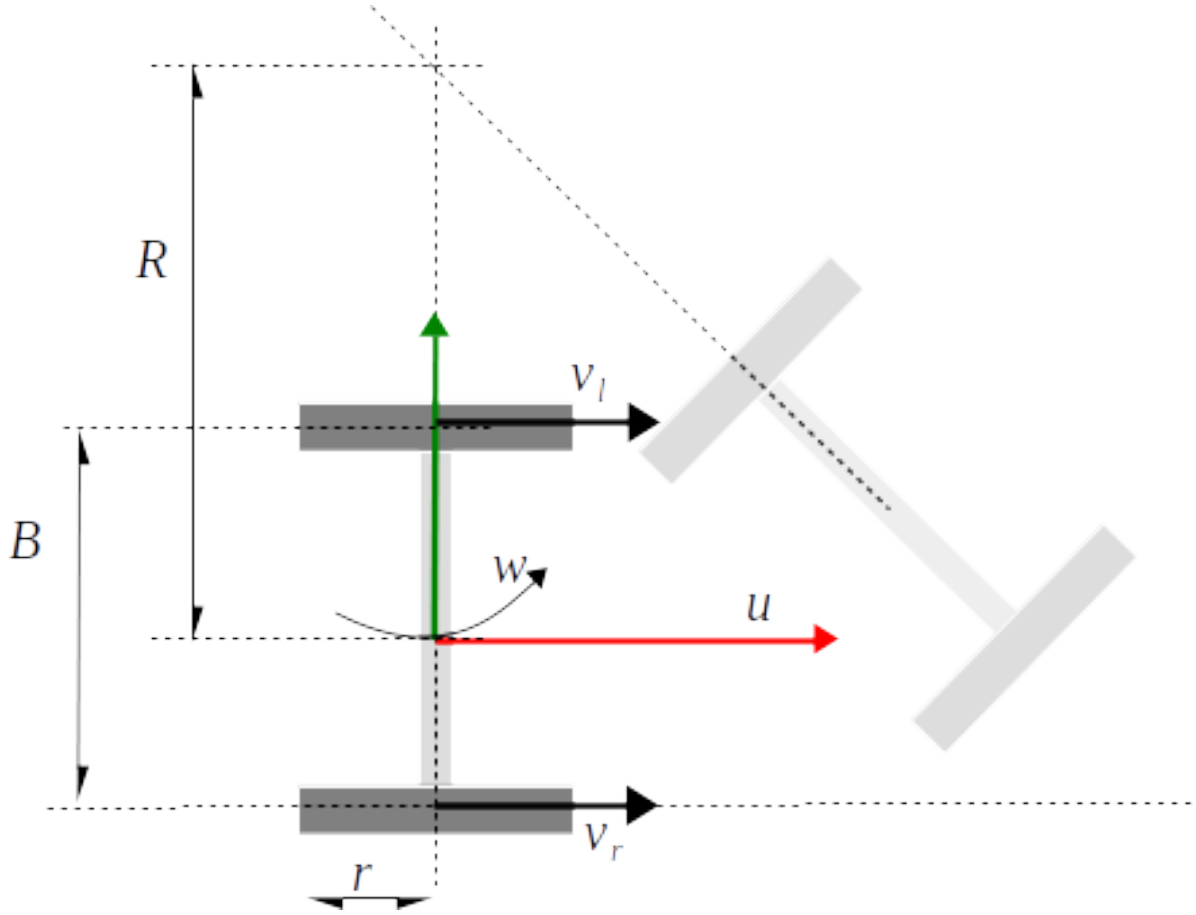

Figure 8: Two wheels differential drive configuration.

Actuator inputs are left and right wheel rotation rates, Ω_l, Ω_r. Constructive parameters are the baseline distance between wheels ,B, and the wheel radius, r. Parameter R in the figure 8 is the curvature radius, which is an intermediate parameter of ineterst.

Placing the platform frame at the mid point on the baseline joining the wheels, we define the twist of this frame as (u, v, w). Applying the Coriolis principle, three equations are deduced, for the left wheel

center point, the right wheel center point and the vehicle origin, respectively:

$$v_l = \Omega_l r = w(R - \frac{B}{2}) \tag{113}$$

$$v_r = \Omega_r r = w(R + \frac{B}{2}) \tag{114}$$

$$u = wR \tag{115}$$

$$\tag{116}$$

operating on $v_r - v_l$, leads to:

$$w = \frac{v_r - v_l}{B} \tag{117}$$

and solving for $v_r + v_l$, the linear forward velocity is:

$$u = \frac{v_r + v_l}{2} \tag{118}$$

In this case the kinematics relation between actuators and platform twist is linear, and can be written in the matrix form:

$$\begin{bmatrix} u \\ w \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{B} & \frac{1}{B} \end{bmatrix} \begin{bmatrix} v_l \\ v_r \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ -\frac{r}{B} & \frac{r}{B} \end{bmatrix} \begin{bmatrix} \Omega_l \\ \Omega_r \end{bmatrix} \tag{119}$$

As in the bicycle case (see subsection 7.5), we might be interested how errors on constructive parameters r and B propagate to the platform twist. In this case the Jacobian of the twist with respect to these parameters is computed as:

$$\mathbf{J}_{rB} = \begin{bmatrix} \frac{1}{2}(\Omega_r + \Omega_l) & 0 \\ \frac{1}{B}(\Omega_r - \Omega_l) & \frac{r}{B^2}(\Omega_l - \Omega_r) \end{bmatrix}; \tag{120}$$

Given an uncertainty in the r and B parameters, represented with a covariance matrix as:

$$\mathbf{C}_{rB} = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_B^2 \end{bmatrix}; \tag{121}$$

the propagation of such uncertainty to twist space is:

$$\mathbf{C}_{uw} = \mathbf{J}_{rB}\mathbf{C}_{rB}\mathbf{J}_{rB}^T = \begin{bmatrix} \frac{\sigma_r^2}{4}(\Omega_r + \Omega_l)^2 & \frac{\sigma_r^2}{4}(\Omega_r + \Omega_l)(\Omega_r - \Omega_l) \\ \frac{\sigma_r^2}{4}(\Omega_r + \Omega_l)(\Omega_r - \Omega_l) & \frac{\sigma_r^2}{4}(\Omega_r - \Omega_l)^2 + \frac{\sigma_B^2 r^2}{B^4}(\Omega_l - \Omega_r)^2 \end{bmatrix}; \tag{122}$$

Result in equation 122 suggests that error in baseline length B only affects the uncertainty of the platform rotation rate, while errors in wheel radius r propagates to both components of the twist u and w.

Inverse kinematics Inverse kinematics for the two wheels differential configuration can be derived by inverting the linear model in equation 119, or from equations 118 and 117:

$$v_r = u + w\frac{B}{2} \tag{123}$$

$$v_l = u - w\frac{B}{2} \tag{124}$$

$$\tag{125}$$

which suggests that for positive rotations, the right wheel should turn faster than the left one.

7.8 4 wheels differential drive

At kinematics level, this configuration reduces to the case of two wheel differential drive since both left wheels have the same rotation rate and both right wheels have also the same rotation rate. However, when the vehicle is rotating, null lateral velocity at wheels is no longer fulfilled, so the wheels slip when turning.

7.9 Ackermann

Ackermann system is the standard steering for most of the nowadays cars and trucks. Figure 9 shows the steering mechanism, the constructive parameters B and L and the input actuators α and u, the later being directly the forward velocity to be transmitted to each rear wheel.

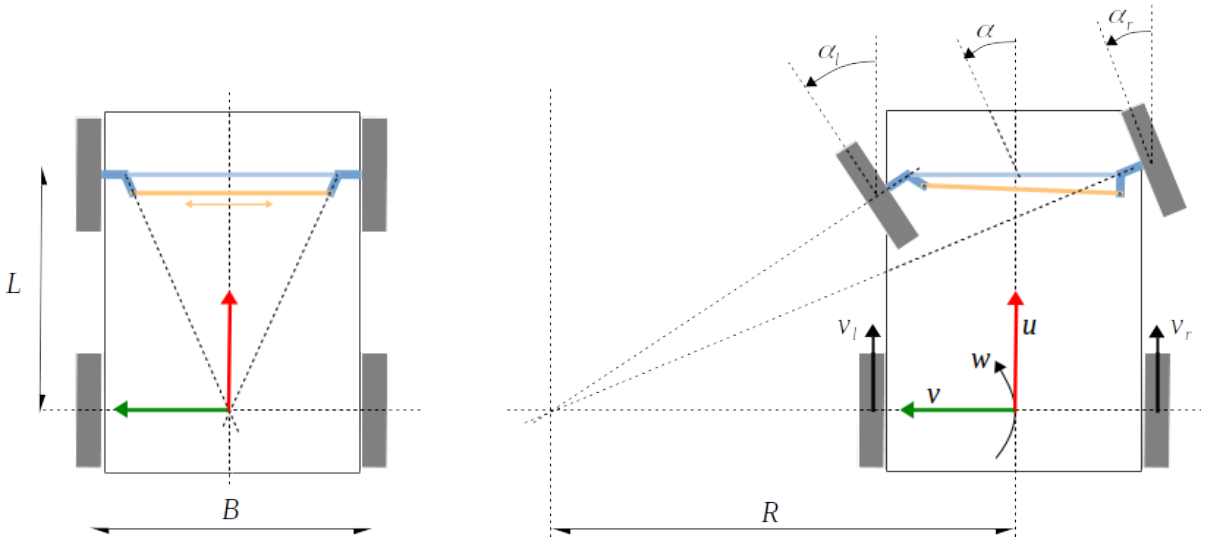

Figure 9: Ackermann configuration. Steering is achieved by actuating the orange bar.

The kinematics of the Ackermann configuration is reduced to the kinematics of the bike seen at subsection 7.5, with input actuators α and u, and constructive parameter L.

However, the Ackermann mechanism ensures a relation between the three steering angles α_l, α_r and α. Given an steering angle α, the tangent of the three triangles with vertex at ICR are:

$$\tan\alpha_l = \frac{L}{R - \frac{B}{2}} \tag{126}$$

$$\tan\alpha = \frac{L}{R} \tag{127}$$

$$\tan\alpha_r = \frac{L}{R + \frac{B}{2}} \tag{128}$$

Solving for R at each equation leads to the following relation that Ackemrann steering should fulfill:

$$\tan\alpha_l = \frac{1}{\frac{1}{\tan\alpha} - \frac{B}{2L}} \tag{129}$$

$$\tan\alpha_r = \frac{1}{\frac{1}{\tan\alpha} + \frac{B}{2L}} \tag{130}$$

$$\tag{131}$$

which is of interest in case of electronic Ackermann steering.

Even if forward velocity u is transmitted to each rear wheel, velocities of each of these wheels are different and constrained with the vehicle twist. Usually a differential mechanism implements this difference of speeds, but an electronic implementation may implement that, fulfilling the following:

$$v_l = w(R - \frac{B}{2}) \tag{132}$$

$$v_r = w(R + \frac{B}{2}) \tag{133}$$

which as it has been seen with the two wheel differential drive at equations 123, it leads to:

$$v_l = u - w\frac{B}{2} \tag{134}$$

$$v_r = u + w\frac{B}{2} \tag{135}$$

7.10 Double Ackermann

Double Ackermann configuration refers to those vehicles having an Ackermann steering system (mechanical or electronical) for front and back wheels, actuating both with the same steering angle, so the ICR point is no longer at the line joining rear wheels , but it is at the line parallel to that but passing through the center of the vehicle. This allows such vehicles to make sharper turns than single Ackermann steering configurations. The configuration of double Ackermann steering can be reduced to a bicycle with steering at front and back wheels, as shown in figure 10.

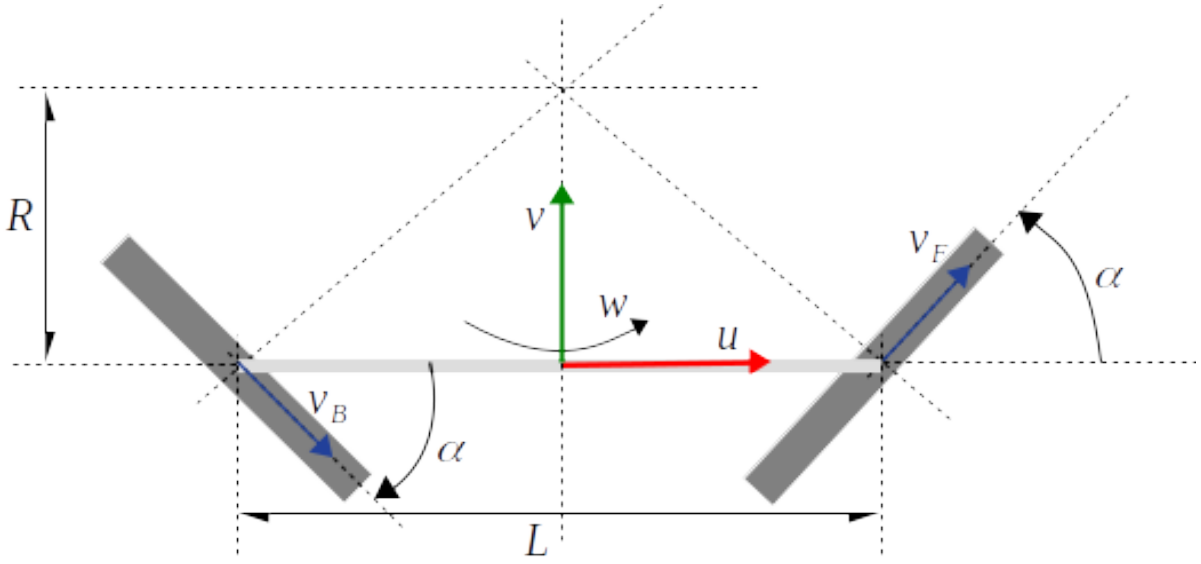

Figure 10: Double Ackermann configuration can be reduced to the double steering bike.

The analysis of this double steering bicycle starts by applying the Coriols principle at the center of

each wheel, as well as at the center of the bicycle:

$$\mathbf{v_B} = \begin{bmatrix} 0 \\ 0 \\ w \end{bmatrix} \times \begin{bmatrix} -\frac{L}{2} \\ -R \\ 0 \end{bmatrix} = \begin{bmatrix} wR \\ -w\frac{L}{2} \\ 0 \end{bmatrix} \tag{136}$$

$$\mathbf{u} = \begin{bmatrix} 0 \\ 0 \\ w \end{bmatrix} \times \begin{bmatrix} 0 \\ -R \\ 0 \end{bmatrix} = \begin{bmatrix} wR \\ 0 \\ 0 \end{bmatrix} \tag{137}$$

$$\mathbf{v_F} = \begin{bmatrix} 0 \\ 0 \\ w \end{bmatrix} \times \begin{bmatrix} \frac{L}{2} \\ -R \\ 0 \end{bmatrix} = \begin{bmatrix} -wR \\ w\frac{L}{2} \\ 0 \end{bmatrix} \tag{138}$$

The above velocity vectors $\mathbf{v_B}$, $\mathbf{u}$ and $\mathbf{v_F}$ are expressed with respect to a frame placed at ICR and aligned with the vehicle frame, so the same velocity vectors are also with respect to the vehicle frame (since there is no rotation between these two frames). For the back wheel, in 2D, we have:

$$\mathbf{v_B} = \begin{bmatrix} wR \\ -w\frac{L}{2} \end{bmatrix} = |\mathbf{v_B}| \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix} = \Omega_B r \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix} \tag{139}$$

so we can find R and w as:

$$R = \frac{\Omega_B r}{w}\cos\alpha \tag{140}$$

$$w = \frac{2\Omega_B r}{L}\sin\alpha \tag{141}$$

$$\tag{142}$$

and then, with the expression of $\mathbf{u} = [u_x\ u_y]^T$ in 2D:

$$u_x = wR = \Omega_B r\cos\alpha\ ;\ \ u_y = 0; \tag{143}$$

A practical case of double Ackermann steering (or even more than double!) is the straddle carrier family (see figure 11), which are large vehicles that need to manoeuver usually in tight spaces with respect to their dimensions L and B.

7.11 Holonomic wheels

Holonomic wheels refers to those wheels that have an extra set of small rollers mounted with their axis forming an angle γ with the wheel perpendicular direction of driving. This angle γ is usually 0 degrees for *omniwheels* or 45 degrees for *mecanum* wheels. These rollers allow the wheel to freely move in lateral motion, while keeping the actuated forward (standard) motion of a wheel, so the constraint of null lateral motion described in 7.4 no longer fulfills.

Figure12 shows the involved frames and vectors to analyze the inverse kinematics of an omniwheel mounted in a vehicle frame.

Given a vehicle twist $[uvw]$, the procedure will find the inverse kinematics model, so the required wheel angular speeds for each wheel. The first step is to find the linear velocity of the wheel center with respect to the vehicle base frame $\mathscr{F}^B$, given the vehicle twist, by applying the Coriolis law:

$$\mathbf{v_i^B} = \begin{bmatrix} u \\ v \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ w \end{bmatrix} \times \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} u \\ v \\ 0 \end{bmatrix} + \begin{bmatrix} -wy_i \\ wx_i \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -y_i \\ 0 & 1 & x_i \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{144}$$

Figure 11: Straddle carriers are big vehicles that need to perform good manoeuvring, so they have double (or even quadruple like in this picture) Ackermann steering, usually implemented with electronic/software means instead of by a mechanism.

Once we know the wheel center linear speed with respect to the vehicle frame, we will transform it to the wheel frame by applying the rotation (only the 2D part):

$$\mathbf{v_i^W} = \begin{bmatrix} \cos\beta_i & \sin\beta_i \\ -\sin\beta_i & \cos\beta_i \end{bmatrix} \begin{bmatrix} 1 & 0 & -y_i \\ 0 & 1 & x_i \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{145}$$

Finally, from $\mathbf{v_i^W}$, we only take the first component, which is the actuated velocity (aligned with the wheel driving direction):

$$u_i = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \cos\beta_i & \sin\beta_i \\ -\sin\beta_i & \cos\beta_i \end{bmatrix} \begin{bmatrix} 1 & 0 & -y_i \\ 0 & 1 & x_i \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{146}$$

which can be simplified to:

$$u_i = \begin{bmatrix} \cos\beta_i & \sin\beta_i & -y_i\cos\beta_i + x_i\sin\beta_i \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{147}$$

A vehicle with N omniwheels (at least $N = 3$), will stack N equations that constraint the actuated wheel angular speeds given the vehicle twist. The following subsections analyses the cases for $N = 3$ and $N = 4$.

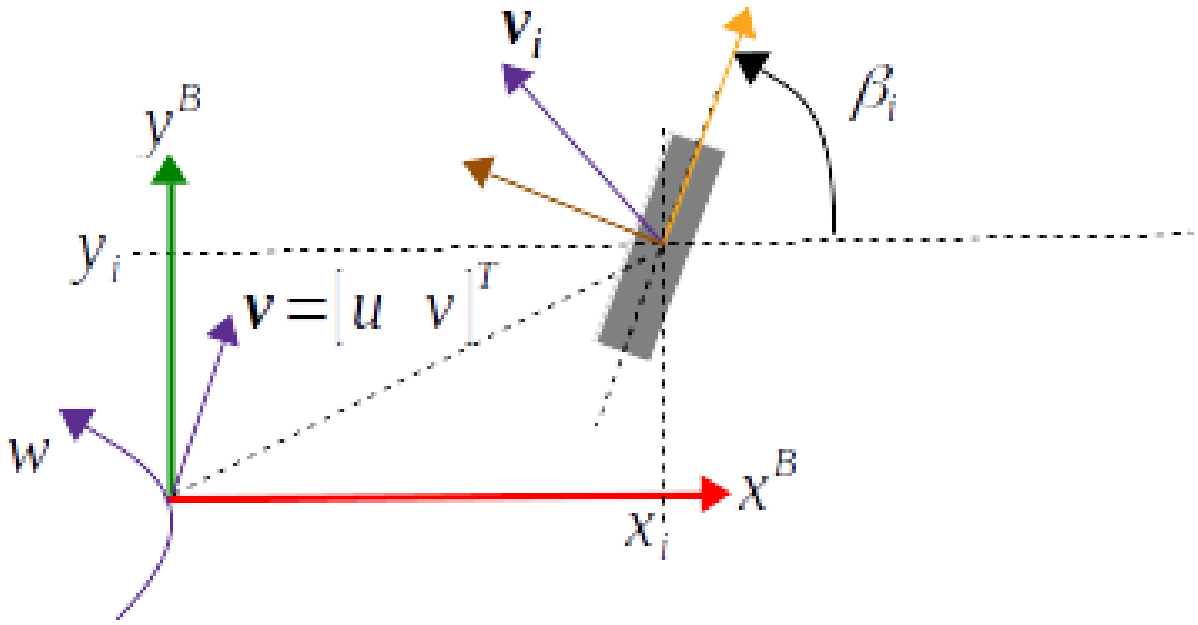

Figure 12: Frames and vectors involved in the omniwheel kinematcs analysis. Vehicle frame in green and red. In yellow wheel driving direction and in brown wheel free sliding direction for omniwheels.

7.12 4 omniwheels

The four omniwheels configuration is a rather standard platform allowing high manoeuvrability and stability. Figure 13 shows this configuration when mounted in a squared platform of side size $2L$.

Table 1 details the constructive parameters x_i, y_i, B_i for this vehicle.

Table 1: Constructive parameters for squared 4-omniwheel vehicle of side size $2L$

	x_i	y_i	β_i
Wheel 0	L	$-L$	$\pi/4$
Wheel 1	L	L	$3\pi/4$
Wheel 2	$-L$	L	$5\pi/4$
Wheel 3	$-L$	$-L$	$7\pi/4$

With the constructive parameters, wheel actuated speeds can be computed by applying equation 147 for each wheel ($i = 0, .., 3$):

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} \cos\beta_0 & \sin\beta_0 & -y_0\cos\beta_0 + x_0\sin\beta_0 \\ \cos\beta_1 & \sin\beta_1 & -y_1\cos\beta_1 + x_1\sin\beta_1 \\ \cos\beta_2 & \sin\beta_2 & -y_2\cos\beta_2 + x_2\sin\beta_2 \\ \cos\beta_3 & \sin\beta_3 & -y_3\cos\beta_3 + x_3\sin\beta_3 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}, \tag{148}$$

so the full inverse kinematics results as follows:

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & L\sqrt{2} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & L\sqrt{2} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & L\sqrt{2} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & L\sqrt{2} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}, \tag{149}$$

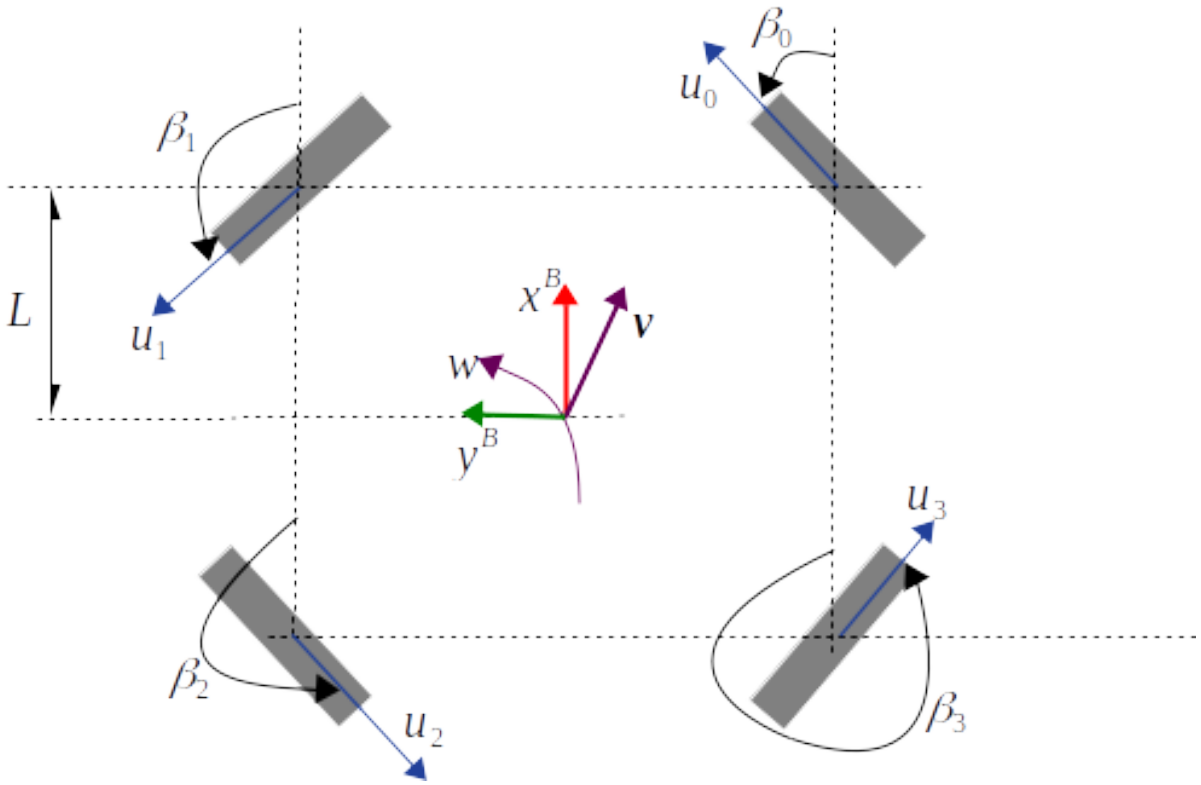

Figure 13: Configuration for a platform with four omniwheels in a square base of side size $2L$.

Please remind that u_i are the linear speeds of wheel centers in the driving direction. To find the angular speed of the wheel we have to apply $\Omega_i = u_i/r$, where r is the radius of the wheel.

7.13 3 omniwheels

A typical configuration using omniwheels is also that of mounting three of them in an equilateral triangle base, as depicted in figure 14.

In this case table 2 shows the constructive parameters x_i, y_i, β_i:

Table 2: Constructive parameters for triangular 3-omniwheel vehicle of side size $2L$

	x_i	y_i	β_i
Wheel 0	L	0	$3\pi/6$
Wheel 1	$-L\sin(\pi/6)$	$L\cos(\pi/6)$	$7\pi/6$
Wheel 2	$-L\sin(\pi/6)$	$-L\cos(\pi/6)$	$11\pi/4$

Therefore, applying equation 147, the inverse kinematics results in the following:

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & L \\ -\frac{\sqrt{3}}{2} & -\frac{1}{2} & L \\ \frac{\sqrt{3}}{2} & -\frac{1}{2} & L \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{150}$$

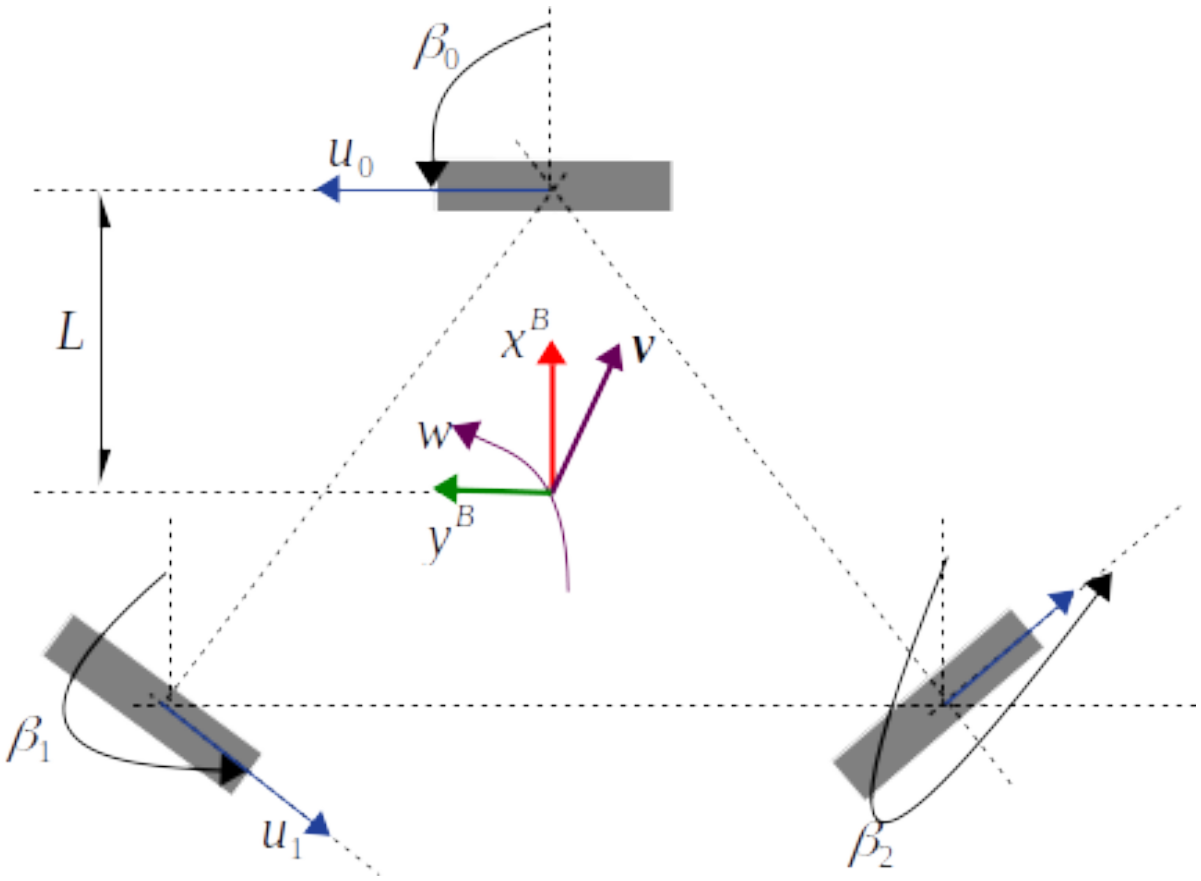

Figure 14: Configuration for a platform with three omniwheels in an equilateral triangle.

7.14 4 mecanum wheels

7.15 Dynamics

Part III
Estimation

8 Least Squares

The first three subsections of this section get inspiration from the book [2], which explains the linear least squares topic in three gradual steps, to finally introduce the Kalman Filter in a natural and smooth way. The same approach is followed in this section, by first presenting the Linear Least Squares, thereafter the Weighted Linear Least Squares and then the Recursive approach. The fourth subsection is devoted to non-linear least squares topic.

The underlying idea and goal of the Least Squares topic is to *find the system state that better explain your measurements*.

8.1 Unweighted Linear Least Squares (UL-LS)

Let be $\mathbf{x} \in \mathbb{R}^n$ the system state of n components, and $\mathbf{z} \in \mathbb{R}^m$ a set of m measurements. Assuming the following linear model for the measurements:

$$\mathbf{z} = \mathbf{H}\mathbf{x} + \mathbf{n}_z \tag{151}$$

The goal of the Least Squares procedure is to estimate the state $\mathbf{x}$ that better explain the measurements $\mathbf{z}$. The problem in the above equation is that the true values of the noise $\mathbf{n}_z$ for each measurement (each component) in $\mathbf{z_i}$ are unknown. However, the noise is assumed to be a random variable following a normal distribution:

$$\mathbf{n}_z \approx \mathcal{N}(0, \sigma_{n_z}^2 \mathbf{I}) \tag{152}$$

so all the measurements are independent and have the same distribution, since the covariance matrix is an identity multiplied by a constant σ_{n_z}. We can define the *expected measurement* vector as:

$$\hat{\mathbf{z}} = \mathbf{H}\mathbf{x} \tag{153}$$

which is the measurements we expect, according the model and given the state $\mathbf{x}$. A measurement error can be also defined as:

$$\mathbf{e}_z = \mathbf{z} - \hat{\mathbf{z}} \tag{154}$$

Given this error, the squared norm of it (squared length) seems to be an appropriate indicator to evaluate how big is this error, so it is defined the following function as the squared norm of the measurement error:

$$f(\mathbf{e}_z) = \mathbf{e}_z^T \mathbf{e}_z = \sum_{i=1}^{m} e_i^2 \tag{155}$$

The criterion to find the state estimate, $\hat{\mathbf{x}}$, that better explains the measurements is, therefore, to find those state $\mathbf{x}$ that minimizes the function $f(\mathbf{e}_z)$, so that minimizes the length of the error vector.

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \ f(\mathbf{e}_z) \tag{156}$$

Developping the expression of $f(\mathbf{e}_z)$ to show explicitly its dependency on $\mathbf{x}$:

$$f(\mathbf{e}_z) = \mathbf{e}_z^T \mathbf{e}_z = (\mathbf{z} - \hat{\mathbf{z}})^T (\mathbf{z} - \hat{\mathbf{z}}) = (\mathbf{z} - \mathbf{H}\mathbf{x})^T (\mathbf{z} - \mathbf{H}\mathbf{x}) = \mathbf{z}^T\mathbf{z} - \mathbf{z}^T\mathbf{H}\mathbf{x} - \mathbf{x}^T\mathbf{H}^T\mathbf{z} + \mathbf{x}^T\mathbf{H}^T\mathbf{H}\mathbf{x} \tag{157}$$

and then, forcing the derivative with respect to $\mathbf{x}$ to be 0 to find the minimum:

$$\frac{\partial f(\mathbf{e}_z)}{\partial \mathbf{x}} = -\mathbf{z}^T\mathbf{H} - \mathbf{z}^T\mathbf{H} + 2\mathbf{x}^T\mathbf{H}^T\mathbf{H} = 0; \ \ \mathbf{x}^T\mathbf{H}^T\mathbf{H} = \mathbf{z}^T\mathbf{H}; \ \ \mathbf{H}^T\mathbf{H}\mathbf{x} = \mathbf{H}^T\mathbf{z}; \tag{158}$$

and solving for $\mathbf{x}$, we finally get the optimal state estimate as:

$$\hat{\mathbf{x}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{z} \tag{159}$$

which provides an estimate of the state, given all the measurements $\mathbf{z}$, a linear measurement model $\mathbf{H}$, according to the criterion of minimal measurement error, and assuming all measurements have independent noise, but they follow the same normal distribution.

8.2 Weighted Linear Least Squares (WL-LS)

In case we have a different confidence level for each measurement. Instead of throwing away the most uncertain measurements, it is wise to also use it, but *weight* them less than the most certain ones. Moreover, some measurements can experience some level of covariance between them. The same linear measurement model than the one used in the unweighted case is assumed here:

$$\mathbf{z} = \mathbf{H}\mathbf{x} + \mathbf{n}_z \tag{160}$$

but in the weighted case, the measurement noise covariance can be a generic covariance matrix:

$$\mathbf{n}_z \approx \mathcal{N}(0, \mathbf{C}_{n_z}) \tag{161}$$

The goal of the Weighted Least Squares procedure is to estimate the state $\mathbf{x}$ that better explain the measurements $\mathbf{z}$, taking into account different confidence levels for each of them.

To derive the final equation, a procedure similar to previous subsection is adopted, but in this case the function $f(\mathbf{e}_z)$ to be minimized is the weighted norm, which can be interpreted as the Mahalanobis distance of the measurement error vector $\mathbf{e}_z$ to the origin:

$$f(\mathbf{e}_z) = \mathbf{e}_z^T \mathbf{C}_{n_z}^{-1} \mathbf{e}_z \tag{162}$$

Imposing the same criterion of minimizing function $f(\mathbf{e}_z)$ with respect to $\mathbf{x}$, we get:

$$\frac{\partial f(\mathbf{e}_z)}{\partial \mathbf{x}} = -\mathbf{z}^T \mathbf{C}_{n_z}^{-1} \mathbf{H} + \mathbf{x}^T \mathbf{H}^T \mathbf{C}_{n_z}^{-1} \mathbf{H} = 0; \tag{163}$$

and solving, the optimal state estimate is finally computed as:

$$\hat{\mathbf{x}} = (\mathbf{H}^T \mathbf{C}_{n_z}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \mathbf{C}_{n_z}^{-1} \mathbf{z} \tag{164}$$

8.3 Recursive Weighted Linear Least Squares (RWL-LS)

The two previous subsections compute the state estimate $\hat{\mathbf{x}}$ once all the measurements were available. This is usually called *batch* mode, when you solve a problem off-line with all the measurements took into account in one single step. But real robotic estimation processes usually receive measurements one by one, since they usually come from some sensor source or detector process. Solving at each iteration the entire problem from the beginning would produce huge measurement vectors $\mathbf{z}$ after a while, but even more critical, untreatable covariance matrices $\mathbf{H}$ and $\mathbf{C}_{n_z}$ to be inverted. So it is important to find a recursive approach to solve a linear least squares in such context.

At a given iteration t, we continue to assume a linear measurement model, which *may* change at each iteration, as it is indicated with the ^{t} superindex:

$$\mathbf{z}^t = \mathbf{H}^t \mathbf{x}^t + \mathbf{n}_z^t, \tag{165}$$

but now, a linear system model is also proposed which recursively updates the current state, based on the previous one:

$$\mathbf{x}^t = \mathbf{x}^{t-1} + \mathbf{K}^t (\mathbf{z}^t - \mathbf{H}^t \mathbf{x}^{t-1}). \tag{166}$$

Given an initialization of $\mathbf{x}^0$, right-sides of both equations above are known, exceptuating the matrix $\mathbf{K}^t$, called the *gain*, and the particular values of the mesurement noise in a given iteration, $\mathbf{n}_z^t$. However the noise distribution is assumed to be known, as we did in the non-recursive case. In order to derive the final set of equations for the recursive algorithm, the goal is to find this gain matrix $\mathbf{K}^t$. In that case,

the criterion will change, and it will be to minimize the sum of the variances of the estimation error at iteration t. So the function to minimize is:

$$f(\mathbf{e}_x^t) = Tr\{\mathbf{C}_x^t\}, \tag{167}$$

where the operator $Tr\{\}$ is the trace of the matrix, which is the sum of all diagonal elements (variances) of the covariance matrix. Expressing the estimation error as:

$$\mathbf{e}_x^t = \mathbf{x}^t - \hat{\mathbf{x}}^t \tag{168}$$

so the covariance matrix, by definition is the expectation of $\mathbf{e}_x\mathbf{e}_x^T$:

$$\mathbf{C}_x^t = \mathscr{E}\{\mathbf{e}_x^t(\mathbf{e}_x^t)^T\} \tag{169}$$

which after some maths (see [2]), leads to:

$$\mathbf{C}_x^t = (\mathbf{I} - \mathbf{K}^t\mathbf{H}^t)\mathbf{C}_x^{t-1}(\mathbf{I} - \mathbf{K}^t\mathbf{H}^t)^T + \mathbf{K}^t\mathbf{C}_{n_z}^t(\mathbf{K}^t)^T \tag{170}$$

Coming back to our minimization goal,

$$f(\mathbf{e}_x^t) = Tr\{\mathbf{C}_x^t\} \rightarrow \frac{\partial f(\mathbf{e}_x^t)}{\partial \mathbf{K}^t} = 0 \tag{171}$$

which finishes with the expression for the gain:

$$\mathbf{K}^t = \mathbf{C}_x^{t-1}(\mathbf{H}^t)^T(\mathbf{H}^t\mathbf{C}_x^{t-1}(\mathbf{H}^t)^T + \mathbf{C}_{n_z}^t)^{-1} \tag{172}$$

The algorithm 1 summarizes all the steps for the recursive least squares approach. The algorithm requires three inputs to start looping: the initial guess for the state estimate, $\mathbf{x}^0$, the initial guess for the covariance of the state estimate, $\mathbf{C}_x^0$, and the measurement model, $\mathbf{H}^t$ which may, or may not, be time dependent. The algorithm provides at the end of each iteration t the state estimate that better explain the measurements received up to t, according the criterion of minimizing the trace of $\mathbf{C}_x^t$.

Algorithm 1 Recursive Weighted Linear Least Squares

INPUTS: $\hat{\mathbf{x}}^0, \mathbf{C}_x^0, \mathbf{H}^t, \mathbf{C}_{n_z}^t$
OUTPUT: $\hat{\mathbf{x}}^t, \mathbf{C}_x^t$, at each iteration
  INIT: $\hat{\mathbf{x}}^0, \mathbf{C}_x^0$
  FOR EACH ITERATION
    $\mathbf{H}^t$ //Compute it, if it is not constant
    $\hat{\mathbf{z}}^t = \mathbf{H}^t\hat{\mathbf{x}}^{t-1}$ //Compute the expected measurement
    $\mathbf{K}^t = \mathbf{C}_x^{t-1}(\mathbf{H}^t)^T(\mathbf{H}^t\mathbf{C}_x^{t-1}(\mathbf{H}^t)^T + \mathbf{C}_{n_z}^t)^{-1}$ //Compute the gain
    $\hat{\mathbf{x}}^t = \hat{\mathbf{x}}^{t-1} + \mathbf{K}^t(\mathbf{z}^t - \hat{\mathbf{z}}^t)$ //Update the state estimate
    $\mathbf{C}_x^t = (\mathbf{I} - \mathbf{K}^t\mathbf{H}^t)\mathbf{C}_x^{t-1}(\mathbf{I} - \mathbf{K}^t\mathbf{H}^t)^T + \mathbf{K}^t\mathbf{C}_{n_z}^t(\mathbf{K}^t)^T$ //Update the covariance of the state estimate
  return $\hat{\mathbf{x}}^t, \mathbf{C}_x^t$
  END FOR

Example 9. Estimate a constant velocity from wheel encoders. This example assumes a 4-wheeled vehicle running at constant forward velocity v and rotational rate ω on a plane. We want to estimate these two variables, from the measurements we get from each wheel encoder. So, the main step we have to solve is to find the measurement model, which relates the state $\mathbf{x} = [v\ \omega]$ with these measurements.

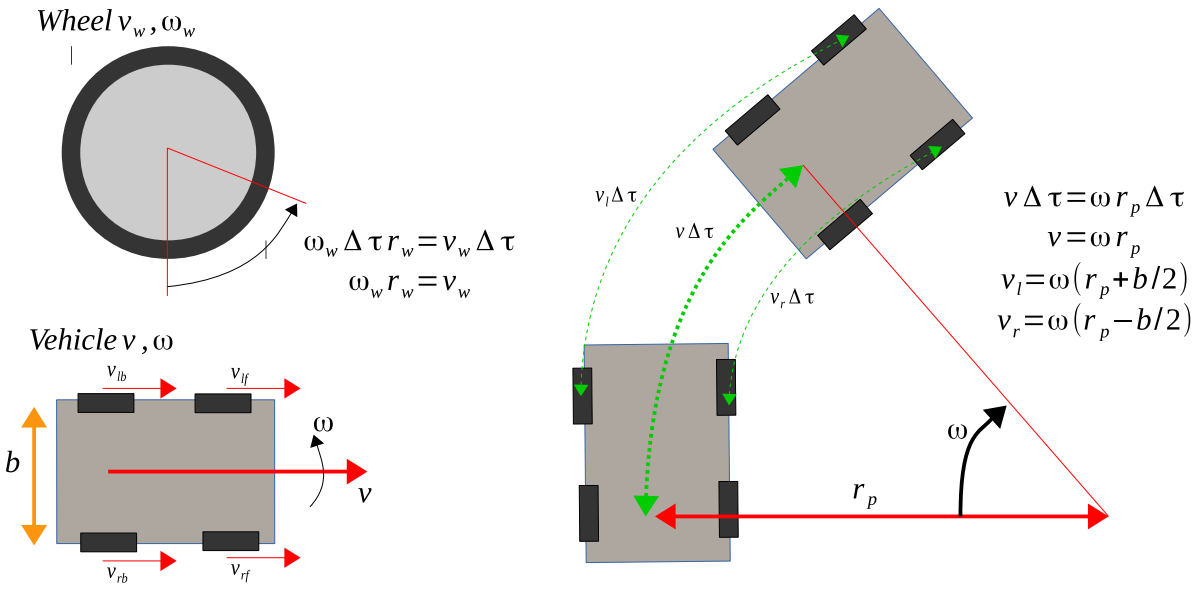

Figure 15: Differential 4-wheel kinematics: from wheel rotational rates to vehicle linear and rotational velocities v, ω.

The measurements we have are the four linear velocities provided by the four encoders mounted at each wheel axis (see figure 15). These linear velocities are directly the rotational velocities, ω_w reported by each encoder multiplied by the wheel radius, r_w:

$$v_{lf} = \omega_{lf} r_{lf};\ v_{lb} = \omega_{lb} r_{lb};\ v_{rf} = \omega_{rf} r_{rf};\ v_{rb} = \omega_{rb} r_{rb} \tag{173}$$

where ω_{**}, v_{**} and r_{**} are rotational and linear speeds and radius for each wheel (left/right front/back). We consider the radius of the wheels to be precisely known, r_w, so the only random variable is each rotational speed measurement, which leads to random variables at linear speed for each wheel.

From the kinematics of a 4-wheeled platform (Figure 15), we have the following two equations, for left and right velocities, according to the rotational rate of the vehicle. These equations relate the arc run by the robot, for the left-wheels and for the right ones, during a $\Delta\tau$ time step:

$$\begin{aligned} v_l \Delta\tau = &\quad \omega \Delta\tau (r_c + \tfrac{b}{2}) \\ v_r \Delta\tau = &\quad \omega \Delta\tau (r_c - \tfrac{b}{2}) \end{aligned} \tag{174}$$

where the curvature radius of the vehicle's trajectory, r_c, is provided by the relation $v = \omega r_c$. Rearranging terms, we finish with the following two equations which explain measurements v_{l*}, v_{r*} as a function of the system state $\mathbf{x} = [v\ \omega]^T$

$$\begin{aligned} v_{l*} &= v + \omega \tfrac{b}{2} \\ v_{r*} &= v - \omega \tfrac{b}{2} \end{aligned} \tag{175}$$

so the measurement process is modelled as:

$$\mathbf{z}^t = \mathbf{H}^t \mathbf{x}^t + \mathbf{n}_z^t \tag{176}$$

where each term of the above equation is:

$$\mathbf{z}^t = \begin{bmatrix} v_{lf} \\ v_{lb} \\ v_{rf} \\ v_{rb} \end{bmatrix};\ \mathbf{H}^t = \mathbf{H} = \begin{bmatrix} 1 & b/2 \\ 1 & b/2 \\ 1 & -b/2 \\ 1 & -b/2 \end{bmatrix};\ \mathbf{x}^t = \begin{bmatrix} v \\ \omega \end{bmatrix}^t;\ \mathbf{n}_z^t \sim \mathcal{N}(\mathbf{0}, \mathbf{C}_{n_z}) \tag{177}$$

At this point, we need to model the measurement noise, by setting the coefficients of the $\mathbf{C}_{n_z}$ matrix according of data and knowledge we may have coming from the device datasheets and/or the measurement procedures (misscalibrations, missynchronizations, noise, unmodelled vehicle slippages, ...). To illustrate better the effect of different uncertainty levels on measurement, the example propose to consider that the front encoders are cheaper than the back ones. With all this in mind, we consider $\sigma_{.f} = 5cm/s^2$ and $\sigma_{.b} = 1cm/s$, and we set covariance terms to $0.5^2 cm^2/s$, so the final measurement covariance matrix is (constant in time):

$$\mathbf{C}_{n_z} = \begin{bmatrix} 0.05^2 & 0.005^2 & 0.005^2 & 0.005^2 \\ 0.005^2 & 0.01^2 & 0.005^2 & 0.005^2 \\ 0.005^2 & 0.005^2 & 0.05^2 & 0.005^2 \\ 0.005^2 & 0.005^2 & 0.005^2 & 0.01^2 \end{bmatrix} \tag{178}$$

Finally, we need the initial guess for the state estimate and state covariance. The initial guess for the state can be set by some knowledge about the system, or may be by collecting a short initial subset of measurements, computing the mean of them and solving the expression:

$$\begin{aligned} v &= \frac{v_r + v_l}{2} \\ \omega &= \frac{v_l - v_r}{b} \end{aligned} \tag{179}$$

which comes from adding and substracting the two equations in 174. Fo state covariance matrix, the initial guess have to be enough *noisy* to allow the algorithm to take effect from measurements appropiately, so it could be set to:

$$\mathbf{C}_x^0 = \begin{bmatrix} 1^2 & 0 \\ 0 & 0.2 \end{bmatrix} \tag{180}$$

which indicates $\sqrt{1} m/s$ in linear standard deviation and $\sqrt{0.2} rad/s$ in rotational rate standard deviation.

If we don't have actual measurements from a set of encoders, to simulate the measurement set is required to generate them syntethically, taking into account some stathistics. This is performed at the first lines of the SciLab code below. The code continues initiliazing the involved matrixes, and then it goes to the main loop that solves iteratively the least squares estimate for v and ω.

```
//clear
clear;

//user entries
n_iter = 200; //number of iterations
bb = 2; //vehicle axis width
v_true = 2.17; //true linear vehicle speed[m/s]
w_true = 0.12; //true rotational vehicle speed [rad/s]

//above vehicle linear and rot speeds, causes true left and right wheel linear speeds
v_l_true = v_true + w_true*bb/2;
v_r_true = v_true - w_true*bb/2;

//Generate noisy measurements, for each of the four encoders, at each iteration
//Measurements are each wheel linear velocity, arranged as: z=[v_lf;v_lb;v_rf;v_rb] (l:left, r:right, f:front, b:back)
sigma_v_f = 0.05; //std dev of linear wheel speed, for front encoders [m/s]. They are cheaper than back ones!
sigma_v_b = 0.01; //std dev of linear wheel speed, for back encoders [m/s]. They are better than front ones!
rand("normal");//set the distribution type to the generator
v_lf = v_l_true + rand(1,n_iter)*sigma_v_f;//left front measurements
v_lb = v_l_true + rand(1,n_iter)*sigma_v_b;//left back measurements
v_rf = v_r_true + rand(1,n_iter)*sigma_v_f;//right front measurements
v_rb = v_r_true + rand(1,n_iter)*sigma_v_b;//right back measurements
z = [v_lf;v_lb;v_rf;v_rb]; //stack all measurements in a single matrix

//****************** Recursive Least Squares starts here. ***********************
//initial guess (state mean and covariance)
x_est = [0;0]; //for the state: [v w]
C_x = [1 0; 0 0.2]; //for the state covariance

//measurement model (constant)
H = [1 bb/2; 1 bb/2; 1 -bb/2; 1 -bb/2;]

//sets measurement process covariance (use values of sigma previously used to generate data)
sigma2_v_fb = 0.005^2; //covariance front-back linear wheel speeds [m^2/s^2]
C_nz = [sigma_v_f^2 sigma2_v_fb sigma2_v_fb sigma2_v_fb; sigma2_v_fb sigma_v_b^2 sigma2_v_fb sigma2_v_fb;
          sigma2_v_fb sigma2_v_fb sigma_v_f^2 sigma2_v_fb; sigma2_v_fb sigma2_v_fb sigma2_v_fb sigma_v_b^2];

//init other usefuls matrixes
I = eye(2,2); //set a 2x2 Identity
x_est_array = []; //state estimate at each iteration
```

```
tr_C_x_array = []; //trace of state covariance matrix

//recursive loop
for ii=1:n_iter
    //compute recursive weighted linear least squares
    z_exp = H*x_est; //expected measurememt
    K = C_x*H'*inv(H*C_x*H'+C_nz); //gain
    x_est = x_est + K*(z(:,ii)-z_exp); //update state estimate
    C_x = (I-K*H)*C_x*(I-K*H)' + K*C_nz*K'; //update state covariance

    //collect results
    x_est_array = [x_est_array x_est];
    tr_C_x_array = [tr_C_x_array trace(C_x)];
end
//****************** Recursive Least Squares ends here. ***********************

//plots
figure('BackgroundColor',[1 1 1]);
plot(x_est_array(1,:));
ph = gca(); // handle
ph.x_label.text = 'iteration';
ph.y_label.text = 'Linear Velocity, v[m/s]';
ph.axes_visible = ["on","on","off"]
ph.grid = [1,1];
ph.auto_scale="on";
figure('BackgroundColor',[1 1 1]);
plot(x_est_array(2,:));
ph = gca(); // handle
ph.x_label.text = 'iteration';
ph.y_label.text = 'Rotational Velocity, w[rad/s]';
ph.axes_visible = ["on","on","off"]
ph.grid = [1,1];
ph.auto_scale="on";
figure('BackgroundColor',[1 1 1]);
plot(tr_C_x_array);
ph = gca(); // handle
ph.x_label.text = 'iteration';
ph.y_label.text = 'Trace(C_x)';
ph.axes_visible = ["on","on","off"]
ph.grid = [1,1];
ph.auto_scale="on";
```

Figure 16 shows the evolution of each component of the state estimate $\hat{\mathbf{x}} = [\hat{v}\ \hat{\omega}]$, while the recursive algorithm iterates, as well as how the trace of the covariance matrix $\mathbf{C}_x^t$ drops, indicating that the output state estimate decreases its uncertainty.

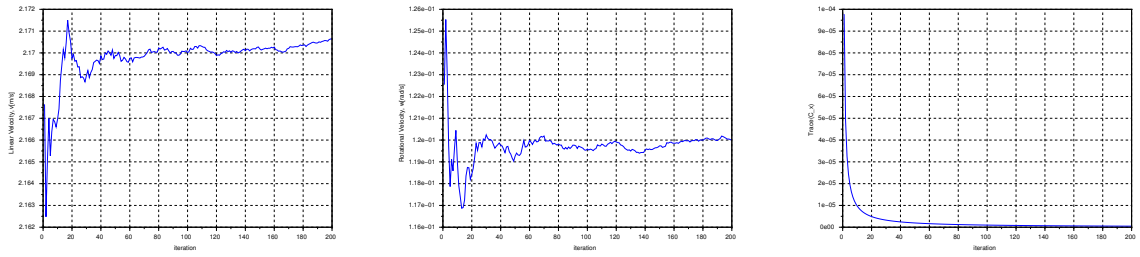

Figure 16: Evolution of linear (left) and rotational (center) speeds, and trace of covariance matrix (right), while the algorithm iterates.

8.4 Linear Least Squares on homeogeneous systems (LS-HS)

Sometimes the problem to solve has different starting situation:

$$\mathbf{A}\mathbf{x} = \mathbf{0} \tag{181}$$

where $\mathbf{x}$ is the state vector (unknown parameters) and the matrix $\mathbf{A}$ contains the measurement data. A typical case is when estimating the parameters of an homogeneous line or plane, given a set of data points. beyond the trivial solution which is the null vectos, the lesat squares solution of such systems of equations is computed as follows through a SVD decomposition of matrix $\mathbf{A}$:

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T \tag{182}$$

And the result is in the last column of matrix **V**. More details on SVD can be found at subsection 4.2) of this document.

8.5 Non-Linear Least Squares (NL-LS)

9 Kalman Filter

Kalman Filter can be interpreted as an extension of the recursive least squares seen at subsection 8.3, for those cases where the state of the system we want to estimate changes over the time, so it has a certain known dynamics, so the algorithm can take benefit of this knowledge of the system motion.

This chapter firstly introduces the basic version of the Kalman Filter, and thereafter explains other useful extensions such as the Extended Kalman Filter and the Error State Kalman Filter. The main bibliographic source of inspiration was the book in [2].

9.1 Kalman Filter (KF)

Given a dynamic system whose states we want to estimate:

$$\mathbf{x}^t = \mathbf{F}^t \mathbf{x}^{t-1} + \mathbf{G}^t \mathbf{u}^t + \mathbf{n}_x^t \tag{183}$$

$$\mathbf{z}^t = \mathbf{H}^t \mathbf{x}^t + \mathbf{n}_z^t \tag{184}$$

with

$$\mathbf{n}_x^t \approx \mathcal{N}(0, \mathbf{C}_{n_x}^t), \;\; \mathbf{n}_z^t \approx \mathcal{N}(0, \mathbf{C}_{n_z}^t) \tag{185}$$

so both the system dynamics (first equation) and measurement model (second equation) are considered random processes, with a certain degree of uncertainty modelled by the two Gaussian variables $\mathbf{n}_x^t$ and $\mathbf{n}_z^t$. This uncertainty comes not only from typical noise, but also from unmodelled effects, such as non-linearities not taken into account in such a linear description of the system.

We define two types of state estimates. The *prior* estimate, also called *prediction*, is the estimate considering measurements up to the last iteration and one step of the dynamic model. It is indicated by underlinning the t superindex:

$$\hat{\mathbf{x}}^{\underline{t}} = \mathscr{E}\{\mathbf{x}^t \mid \mathbf{z}^1, \mathbf{z}^2, \ldots \mathbf{z}^{t-1}, \} \tag{186}$$

and the *posterior* estimate, also called *correction*, is the estimate taken into account the current measurement $\mathbf{z}^t$, so the prediction gets corrected by it:

$$\hat{\mathbf{x}}^t = \mathscr{E}\{\mathbf{x}^t \mid \mathbf{z}^1, \mathbf{z}^2, \ldots \mathbf{z}^t, \} \tag{187}$$

Algorithm 2 summarizes the Kalman Filter, where iterations run over a predicition step and a correction one.

In the **prediction step**, the prior estimate is computed, $\hat{\mathbf{x}}^{\underline{t}}$, by making a prediction of the state one time-lapse further, thanks to consider the last posterior estimate, $\hat{\mathbf{x}}^{t-1}$, the known dynamics, $\mathbf{F}^t$, and possible control inputs to the system, $\mathbf{G}^t$ and $\mathbf{u}^t$. In the other hand, the **correction step** uses the current measurement to improve the prediction, following the same approach used in the recursive least squares (see subsection 8.3), but computing the expected measurement, $\hat{\mathbf{z}}^t$, at the state point found in the prediction step.

Depending on the application, this alternate between prediction and correction steps can be modified. For instance, several prediction steps could be computed between two corrections. In multi-sensor cases, some correction steps may involve only a partial subset of measurements, since it is common that not all sensors arrive in the Filter CPU at a same rate. Knowing that the fundamentals of the algorithm are the recursive least squares and the uncertainty propagation, different combinations of prediction and correction steps are completely correct from a theoretical point of view, and can provide practical benefits.

Algorithm 2 Kalman Filter

INPUTS: $\hat{\mathbf{x}}^0, \mathbf{C}_x^0, \mathbf{F}^t, \mathbf{G}^t, \mathbf{H}^t, \mathbf{C}_{n_x}^t, \mathbf{C}_{n_z}^t$
OUTPUT: $\hat{\mathbf{x}}^t, \mathbf{C}_x^t$, at each iteration
FOR EACH ITERATION
PREDICTION
$\mathbf{F}^t, \mathbf{G}^t$ //Compute them if they are not constant
$\hat{\mathbf{x}}_{\bar{}}^t = \mathbf{F}^t \mathbf{x}^{t-1} + \mathbf{G}^t \mathbf{u}^t$ //Prior state estimate
$\mathbf{C}_{\bar{x}}^t = \mathbf{F}^t \mathbf{C}_x^{t-1} (\mathbf{F}^t)^T + \mathbf{C}_{n_x}^t$ //Prior state covariance
CORRECTION
$\mathbf{H}^t$ //Compute it, if it is not constant
$\hat{\mathbf{z}}^t = \mathbf{H}^t \hat{\mathbf{x}}_{\bar{}}^t$ //Compute the expected measurement
$\mathbf{K}^t = \mathbf{C}_{\bar{x}}^t (\mathbf{H}^t)^T (\mathbf{H}^t \mathbf{C}_{\bar{x}}^t (\mathbf{H}^t)^T + \mathbf{C}_{n_z}^t)^{-1}$ //Compute the gain
$\hat{\mathbf{x}}^t = \hat{\mathbf{x}}_{\bar{}}^t + \mathbf{K}^t (\mathbf{z}^t - \hat{\mathbf{z}}^t)$ //Posterior state estimate
$\mathbf{C}_x^t = (\mathbf{I} - \mathbf{K}^t \mathbf{H}^t) \mathbf{C}_{\bar{x}}^t (\mathbf{I} - \mathbf{K}^t \mathbf{H}^t)^T + \mathbf{K}^t \mathbf{C}_{n_z}^t (\mathbf{K}^t)^T$ //Update the covariance of the state estimate
return $\hat{\mathbf{x}}^t, \mathbf{C}_x^t$
END FOR

Observabililty Analysis Observability is a property of a linear system, such as the one defined in equations 183 and 184, that indicates if a state can be estimated/observed. It is important to check for the observability before starting to code an implementation of the Kalman Filter. The following procedure provides a way to perform this check. Given a system state of n dimensions, the observability matrix is mounted as:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{H} \\ \mathbf{HF} \\ \mathbf{HF}^2 \\ \vdots \\ \mathbf{HF}^{n-1} \end{bmatrix} \tag{188}$$

The system is observable if and only of the matrix $\mathbf{Q}$ fulfills $rank(\mathbf{Q}) = n$. Observability is the concept allowing us to estimate states that we do not directly measure, we just *observe*.

Example 10. Object tracking in an image. Object detectors, such as ball or face detectors, typically provide pixel coordinate of the center of the target object at a given rate. However, they usually have false positive and false negative events, the first report detections when no target is present, the later not providing detections when the target is present in the scene. Therefore, it is always interesting to combine the output of a detector with a tracker, since the output of a tracking process provides smoother and more robust values of the object's position in the image.

Given a detector providing detections as object positions in image pixel coordinates, we consider, at iteration t, this detection as the mesaurement:

$$\mathbf{z}^t = [z_x^t \;\; z_y^t]^T \,, \tag{189}$$

and we define the state as:

$$\mathbf{x}^t = [p_x^t \;\; p_y^t \;\; v_x^t \;\; v_y^t]^T . \tag{190}$$

which are the position and the velocity of the object in pixel coordinates. Adding velocities to the state is useful in order to benefit from a dynamic model, even if it is simple, since Kalman filtering allows to exploit this knowledge through prior calculation. So the simple dynamic model we will consider is the

following:

$$\mathbf{x}^t = \mathbf{F}\mathbf{x}^{t-1} + \mathbf{n}_x^t; \quad \mathbf{F}^t = \mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta\tau & 0 \\ 0 & 1 & 0 & \Delta\tau \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad \mathbf{C}_{n_x}^t = \mathbf{C}_{n_x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix}; \tag{191}$$

where $\Delta\tau$ is the time lapse between two iterations in seconds. This dynamic model do not consider control inputs ($\mathbf{u}^t$ in equation 183) since the object is considered to move free, without any control from our system. The uncertainty associated to this dynamic model is characterized as a Gaussian noise, with a covariance matrix set from experimental experience, so it considers 1 pixel of standard deviation in position, 5 pixels of standard deviation in velocity and null covariance terms. Both matrixes $\mathbf{F}$ and $\mathbf{C}_{n_x}$ are considered constant, so the iteration superindex t has been removed. This dynamic model is called *constant velocity* model, since assumes that the objects move thorugh the scene at a constant (linear) velocity. This may not be true, so uncertainty matrix has to be enough large to incorporate these model inaccuracies. This simple motion model may cause poor results, specially if time lapse $\Delta\tau$ is large in comparison with the target speed, but increasing filter rate can mitigate this limitations leading to a better output.

The mesaurement model, which allows to compute the expected measurement from the state is:

$$\mathbf{z}^t = \mathbf{H}\mathbf{x}^t + \mathbf{n}_z^t; \quad \mathbf{H}^t = \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}; \quad \mathbf{C}_{n_z}^t = \mathbf{C}_{n_z} = \begin{bmatrix} 25 & 0 \\ 0 & 25 \end{bmatrix}; \tag{192}$$

which is also a constant-time model, and just expects that the measurement is equal to the pixel position components of the state. The velocity part is not measured, so the matrix $\mathbf{H}$ has only two rows. And the velocity part of the state do not contribute to the measurement, so the zeros in the third and fourth columns of $\mathbf{H}$. Again, a 5 pixels standard deviation is considered to model the innacuracies of the detector.

At this point we summarize the dimensions of the filter. The state space is 4-dimensional ($n = 4$), while the measurement space is 2-dimensional ($m = 2$). Let's check if this design pass the observability test:

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & \Delta\tau & 0 \\ 0 & 1 & 0 & \Delta\tau \\ \vdots & & & \end{bmatrix} \longrightarrow rank(\mathbf{Q}) = 4, \tag{193}$$

so the state is observable.

To start running the filter, we only need to decide how to initialize it. In tracking applications, it is common to initialize the state with the values provided by a first detection, for the pixel position, and null values for the pixel velocities. The state covariance, could be initialized as:

$$\mathbf{C}_x^0 = \begin{bmatrix} 25 & 1 & 0 & 0 \\ 1 & 25 & 0 & 0 \\ 0 & 0 & 100 & 1 \\ 0 & 0 & 1 & 100 \end{bmatrix} \tag{194}$$

Algorithm 3 summarizes the object tracking implementation with the Kalman Filter.

9.2 Extended Kalman Filter (EKF)

9.3 Error State Extended Kalman Filter (ES-EKF)

Algorithm 3 Kalman Filter for Object Tracking

$\hat{\mathbf{x}}^0 \leftarrow [z_x^0\ z_y^0\ 0\ 0]^T$ //init state with first detection
$\mathbf{C}_x^0$ //init state covariance with fixed values
FOR EACH ITERATION
  PREDICTION
    $\hat{\mathbf{x}}_x^t = \mathbf{F}\mathbf{x}^{t-1}$ //Prior state estimate
    $\mathbf{C}_x^t = \mathbf{F}\mathbf{C}_x^{t-1}\mathbf{F}^T + \mathbf{C}_{n_x}$ //Prior state covariance
  CORRECTION
    $\hat{\mathbf{z}}^t = \mathbf{H}\hat{\mathbf{x}}_x^t$ //Compute the expected measurement
    $\mathbf{K}^t = \mathbf{C}_x^t\mathbf{H}^T(\mathbf{H}\mathbf{C}_x^t\mathbf{H}^T + \mathbf{C}_{n_z})^{-1}$ //Compute the gain
    $\hat{\mathbf{x}}^t = \hat{\mathbf{x}}_x^t + \mathbf{K}^t(\mathbf{z}^t - \hat{\mathbf{z}}^t)$ //Posterior state estimate
    $\mathbf{C}_x^t = (\mathbf{I} - \mathbf{K}^t\mathbf{H})\mathbf{C}_x^t(\mathbf{I} - \mathbf{K}^t\mathbf{H})^T + \mathbf{K}^t\mathbf{C}_{n_z}(\mathbf{K}^t)^T$ //Update the covariance of the state estimate
    $\mathbf{C}_x^t \leftarrow \frac{1}{2}(\mathbf{C}_x^t + (\mathbf{C}_x^t)^T)$ //ensures covariance is definite positive
return $\hat{\mathbf{x}}^t, \mathbf{C}_x^t$
END FOR

10 Particle Filter

To do

11 Factor Graphs and Windowed Optimization

To do

12 Odometry

Odometry role, sources and 2D/3D twist integration

13 Map-based Localization

14 Simultaneous Localization and Mapping (SLAM)

Simultaneous localization and mapping is the problem of building a representation of the environment (a map) while keep the robot localized within this representation. It is a problem of major importance in robotics, since mapping is a procedure often required in a deployment step of a mobile robotics system. Moreover, the same techniques developed in SLAM are also used, for instance, to calibrate the position of a set of static markers in the environement, which can be seen as a mapping problem where the map is just built as a set of landmarks.

14.1 Types of maps

A map is a representation of the environment, designed with a target purpose, usually linked with some computational process. Examples of these processes that motivate the design of a map are path planning, compute expected sensor measurements, localize some key points of the environment or share the representation with humans.

Since there is a broad collection of algorithms and sensors to solve the above listed processes, different environment representations are used in robotics. They are mainly classified with two main types: *metric* maps and *topological maps*.

14.1.1 Metric maps

Metric maps are those representations that keep the euclidean spanning of the 2D or 3D space, so the elelments in the map are placed in it with their euclidean coordinates with respect to the map frame. Within metric maps, there are two big families: *dense* maps and *sparse* maps.

Dense maps represent the whole environment, both the free and the occupied space, and do neither extract features nor detect landmarks from them, so areas or volumes are always represented by a set of elements that tesselate the space, basically indicating either their occupancy probability or some other basic data such as color, elevation or surface normal vector. Examples of these representations are occupancy grids for 2D or voxel grids for 3D. These grids can be regular or not, in the later case they can be organized with *N-trees*.

Sparse maps represent the environment as a set of features. These features have euclidean coordinates with respect to the map frame and may also have some descriptive attributes. Examples of these features are geometric (points, lines, planes, corners) or visual (key points such as SURF or ORB). Artificial landmarks (such as QR codes or RF beacons) may be also considered as points in the map, therefore a case of point-based maps where the descriptive attribute is just an *id*.

14.1.2 Topological maps

Topological maps are those representations that describe the environment as a graph, with a set of nodes and links, but metric information is not necessarily kept. Instead, the information represented correspoends to the relationships between nodes. These representations are usually used for high level planning or human-robot interaction, but also for loop closure detections in mapping. Nodes may represent entities such as places, and links may indicate if places are connected so a robot can visit one coming from the other.

14.2 SLAM front-end

According to the choosen map representation, a *front-end* part of the SLAM process is in charge of collecting sensor readings and extract those relevant information to be passed to the back-end, which is the solver part of the problem. Front-end steps are the most tricky part of SLAM, and there is a broad use of techniques. The final success of the whole SLAM proceudre is usually very sensitive to the front-end performance in terms of accuracy, robustness or computational speed.

Examples of processes computed in the front-end may be:

- Geometric feature detection (points, lines, planes, corners).
- Visual feature detection and descriptor.
- Matching 2D/3D between two scans/point clouds, to extract a pose constraint between two points of view of the environment.
- Visual Odomertry between two images, to extract a pose constraint between two points of view of the environment.

Data association in the SLAM front-end, specially for feature-based maps, there is a critical step called *data association* which tries to find correspondences between the set of current detected features and the set of already mapped ones. This step usually uses a first metric check to generate candidate correspondences, and then uses the descriptive attributes of features (detected and already mapped) to finally find correspondences.

14.3 SLAM back-end

The back-end side of the SLAM problem is related to the estimator or solver process to *find the best map according measurements*. In the 1990's and 2000's Kalman filter approaches were proposed and developed, but since 2010's, Graph-based techniques have been developed.

14.4 Kalman Filter based SLAM

The key idea of Kalman filter based SLAM is to augment the robot localization state with the set of coordinates of all the features building the map, so this approach works mainly for sparse maps (feature-based).

In SLAM, the state is:

$$\mathbf{x} = (\mathbf{x}_r^M, \mathbf{x}_0^M, \ldots, \mathbf{x}_i^M, \ldots, \mathbf{x}_{N-1}^M)^T \tag{195}$$

where the first vector $\mathbf{x}_r^M$ is the pose of the robot with respect to the map frame and the other vectors $\mathbf{x}_i^M$ are the coordinates of all the features that build the map. This state is the one to be optimally estimated by the Kalman filter.

In 2D the state becomes to:

$$\mathbf{x} = (x_r^M, y_r^M, \theta_r^M, x_0^M, y_0^M, \ldots, x_i^M, y_i^M, \ldots, x_{N-1}^M, y_{N-1}^M)^T \tag{196}$$

while in 3D, in case of using quaternion representaion for rotations, the state is:

$$\mathbf{x} = (x_r^M, y_r^M, z_r^M, q_i^M, q_j^M, q_k^M, x_0^M, y_0^M, z_0^M, \ldots, x_i^M, y_i^M, z_i^M, \ldots, x_{N-1}^M, y_{N-1}^M, z_{N-1}^M)^T \tag{197}$$

In both cases, the state could be augmented with other robot state components, such as linear or rotational velocities, or accelerations.

Once the state space is defined, the Kalman Filter requires to design two models (see chapter 9): the state prediction model, linearized with the matrix $\mathbf{F}$, and the observation or measurement model, linearized with the matrix $\mathbf{H}$.

We will derive here both models for a differential drive robot on the plane (2D case), and for the landmark-based map, which is, for instance, the case of a map representing the poses of a set of artificial QR marks.

Starting with the **prediction model**, since the poses of the landmarks in the map are assumed to be static, the model is just the predictive model for the vehicle state, and an Identity matrix for the landmarks. The linear predictive model of the vehicle state is computed with the twist input, which is (v,w) for differential platforms:

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & -v\Delta T\sin(\theta_r^M) & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & v\Delta T\cos(\theta_r^M) & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & & \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{F}_r & \mathbf{0}_{3\times 3(N-1)} & \cdots & \cdots & \cdots \\ \mathbf{0}_{3\times 3} & \mathbf{I}_{3\times 3} & \mathbf{0}_{3\times 3(N-2)} & \cdots & \cdots \\ \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{I}_{3\times 3} & \mathbf{0}_{3\times 3(N-3)} & \cdots \\ \vdots & \vdots & & & \vdots \\ \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3(N-2)} & \cdots & \cdots & \mathbf{I}_{3\times 3} \end{bmatrix} \tag{198}$$

where the second expression is the block matrix, very useful in the SLAM formulation since the state size will always depend on the number of landmarks N seen up to the current time.

The $\mathbf{F}_r$ block is the linearization of the odometry pose prediction of a differential drive platform, computed as the time integration of the twist, (v,w):

$$x^{M,t} = x^{M,t-1} + v\Delta T\cos\theta^{M,t-1} \tag{199}$$

$$y^{M,t} = y^{M,t-1} + v\Delta T\sin\theta^{M,t-1} \tag{200}$$

$$\theta^{M,t} = \theta^{M,t-1} + w\Delta T \tag{201}$$

In the other hand, the **measurement model** should relate the SLAM state with the measurement the system expects to get from its perception stage. If the homogeneous transforms of the robot and the i-th mark with respect to the map are $\mathbf{T}_r^M$ and $\mathbf{T}_i^M$ respectively, then the transform of the mark with respect to the robot is $\mathbf{T}_i^r$, which is directly the expected measurement of the i-th mark from the robot point of view:

$$\mathbf{T}_i^M = \mathbf{T}_r^M\mathbf{T}_i^r; \rightarrow \mathbf{T}_i^r = (\mathbf{T}_r^M)^{-1}\mathbf{T}_i^M \tag{202}$$

which expands to (superindex ^{M} is ommited for clarity):

$$\mathbf{T}_i^r = \begin{bmatrix} c\theta_r & s\theta_r & -x_r c\theta_r - y_r s\theta_r \\ -s\theta_r & c\theta_r & x_r s\theta_r - y_r c\theta_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta_i & -s\theta_i & x_i \\ s\theta_i & c\theta_i & y_i \\ 0 & 0 & 1 \end{bmatrix} = \tag{203}$$

$$= \begin{bmatrix} c\theta_r c\theta_i + s\theta_r s\theta_i & -c\theta_r s\theta_i + s\theta_r c\theta_i & x_i c\theta_r + y_i s\theta_r - x_r c\theta_r - y_r s\theta_r \\ -s\theta_r c\theta_i + c\theta_r s\theta_i & s\theta_r s\theta_i + c\theta_r c\theta_i & -x_i s\theta_r + y_i s\theta_r + x_r s\theta_r - y_r c\theta_r \\ 0 & 0 & 1 \end{bmatrix} \tag{204}$$

where, after some arrangements, $\mathbf{T}_i^r$ leads to the following homogeneous transform:

$$\mathbf{T}_i^r = \begin{bmatrix} c(\theta_i - \theta_r) & -s(\theta_i - \theta_r) & (x_i - x_r)c\theta_r + (y_i - y_r)s\theta_r \\ -s(\theta_i - \theta_r) & c(\theta_i - \theta_r) & -(x_i - x_r)s\theta_r + (y_i - y_r)c\theta_r \\ 0 & 0 & 1 \end{bmatrix} \tag{205}$$

so the expected measurement of the i-th mark, given the SLAM state $\mathbf{x}$ is:

$$\hat{\mathbf{z}} = \begin{bmatrix} x_i^r \\ y_i^r \\ \theta_i^r \end{bmatrix} = \begin{bmatrix} (x_i^M - x_r^M)c\theta_r^M + (y_i^M - y_r^M)s\theta_r^M \\ -(x_i^M - x_r^M)s\theta_r^M + (y_i^M - y_r^M)c\theta_r^M \\ \theta_i^M - \theta_r^M \end{bmatrix} \tag{206}$$

Since this expression is not linear with respect to the state variables, to find the matrix $\mathbf{H}$ we have to linearize with respect to state variables:

$$\mathbf{H}_{r,i} = \begin{bmatrix} c\theta_r^M & -s\theta_r^M & -(x_i^M - x_r^M)s\theta_r^M + (y_i^M - y_r^M)c\theta_r^M & \dots & c\theta_r^M & s\theta_r^M & 0 & \dots \\ s\theta_r^M & -c\theta_r^M & -(x_i^M - x_r^M)c\theta_r^M - (y_i^M - y_r^M)s\theta_r^M & \dots & -s\theta_r^M & c\theta_r^M & 0 & \dots \\ 0 & 0 & -1 & \dots & 0 & 0 & 1 & \dots \end{bmatrix} \tag{207}$$

where the left matrix (derivatives with respect to robot state) is $\mathbf{H}_r$, and the right part (derivatives with respect the landmark) is $\mathbf{H}_i$. Therefore the full $\mathbf{H}$ matrix is as follows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_r & \mathbf{H}_0 & \mathbf{0}_{3x3(N-2)} & \dots & \dots \\ \mathbf{H}_r & \mathbf{0}_{3x3} & \mathbf{H}_1 & \mathbf{0}_{3x3(N-3)} & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{H}_r & \mathbf{0}_{3x3(N-2)} & \dots & \dots & \mathbf{H}_{N-1} \end{bmatrix} \tag{208}$$

In such SLAM Kalman filter, the correction step will be performed only for the visible marks. Other marks out of the filed of view of the sensory system will be not corrected. The SLAM state size is dynamic and will grow while new marks are seen by the robot.

14.5 Graph SLAM

15 Multi-View Geometry in Computer Vision

How points and lines change and transform from different projective views, as those taken from a camera.

Part IV
Planning and Control

16 Planning and control

How to plan paths and trajectories at actuator or platform level, and how to control the robot to execute them.

16.1 Spline planning in 1D

Cubic splines in 1D are curves in the t parameter that fulfill zero-order and first-order derivative constraints, so they allow to interpolate between two points in a soft way.

Compute the cubic polynomial in $t \in [0,1]$ domain:

$$x(t) = at^3 + bt^2 + ct + d \tag{209}$$

with the following four constraints:

$$\begin{aligned} x(0) &= x_0 && (210)\\ x(1) &= x_1 && (211)\\ \dot{x}(0) &= \tau_0 && (212)\\ \dot{x}(1) &= \tau_1 && (213)\\ & && (214) \end{aligned}$$

Given that:

$$\dot{x}(t) = 3at^2 + 2bt + c \tag{215}$$

the system of equations to solve is:

$$\begin{aligned} d &= x_0 && (216)\\ a+b+c+d &= x_1 && (217)\\ c &= \tau_0 && (218)\\ 3a+2b+c &= \tau_1 && (219)\\ & && (220) \end{aligned}$$

Substracting 3 times the equation 218 to the equation 220, we obtain:

$$-b-2c-3d = \tau_1 - 3x_1; \rightarrow b = 3(x_1 - x_0) - 2\tau_0 - \tau_1 \tag{221}$$

and solving for a:

$$a = 2(x_0 - x_1) + \tau_0 + \tau_1 \tag{222}$$

so in matrix form we can write:

$$x(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \tau_0 \\ \tau_1 \end{bmatrix} \tag{223}$$

16.2 Spline planning in 2D

16.3 A* for Global Planning

16.4 PID control

The diagram in figure 17 shows the standard situation for a PID controller.

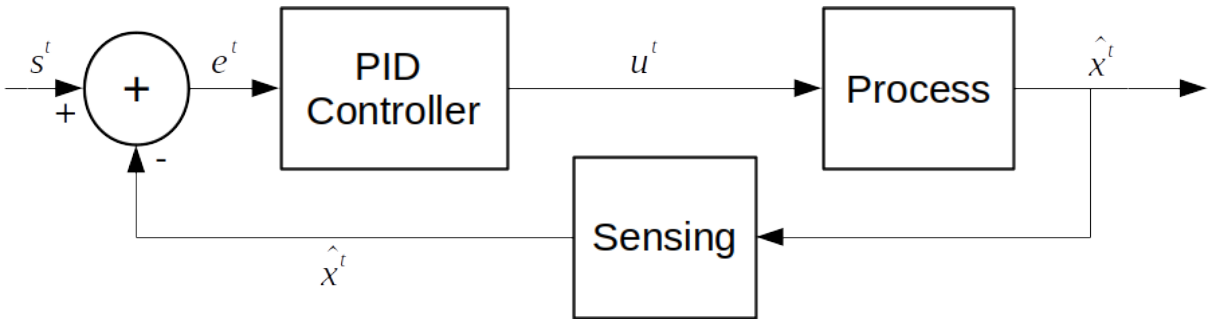

Figure 17: General diagram for a PID controller.

Given a desired, reference or set-point signal s^t, a control command u^t, a process state x^t and a process state estimate $\hat{x}^t$ issued from a sensing/perception stage, the error signal is defined as:

$$e^t = s^t - \hat{x}^t \tag{224}$$

$$\dot{e}^t = \frac{e^t - e^{t-1}}{\Delta T} \tag{225}$$

$$\vec{e}^t = \sum_{j=0}^{N} e^{t-j} \Delta T \tag{226}$$

$$\tag{227}$$

and the output control command is then computed as:

$$u^t = K_p e^t \; + \; K_i \vec{e}^t \; + \; K_d \dot{e}^t \tag{228}$$

Algorithm 4 summarizes the PID controller using pseudocode related to STL C++ vectors.

Algorithm 4 PID controller

```
WHILE (true)
    s = getSetPoint()
    x = getSensing()
    e = s − x
    ed = (e − ev.back())/ΔT
    ev.push_back(e) = (e − ev.back())/ΔT
    IF ev.size() > N
        ei = ei + eΔT − ev.front()ΔT
        ev.pop_front()
    ELSE
        ei = ei + eΔT
    END IF
    u = Kp e + Ki ei + Kd ed
END WHILE
```

The main difficulty of the PID in practical situations is to correctly tune the constants K_p, K_i and K_d. The general guidelines for this tunning are, in order:

1. $K_p = K_i = K_d = 0$
2. Increase K_p up to get some reasonable result.

3. Reduce the overshoot effect by increasing K_d.

4. Last fine tunning by increasing K_i.

The main interpretation of this PID controller is that proportional part is the one mainly driving the control, but the derivative part allows the controller to incorporate some predictive behavior. Finnally the integral contribution gives the possibilty to accurately reach a control reference when the proportional part is near zero.

16.5 Trajectory control

16.6 Dynamical Window Approach (DWA) for wheeled robots

Part V
Implementation Tools and Tips

17 Common Algorithms

RANSAC, ICP, HOUGH Transform, K-means clustering

18 Device communication protocols

EtherCAT, CANopen, EthernetIP, SPI, SSI,

References

[1] H.V. Henderson and S.R. Searle. On deriving the inverse of a sum of matrices. *Biometrics Unit Series*, (BU-647-M), 1980.

[2] D. Simon. *Optimal State Estimation*. John Wiley & Sons, Inc., 2006.