Derivation and Implementation of a Full 6D EKF-based Solution to Bearing-Range SLAM

Jose-Luis Blanco
jlblanco@ctima.uma.es

Technical Report

Perception and Mobile Robots Research Group
University of Málaga, Spain

March 7, 2008

Abstract

This report addresses the formulation and the implementation of a full 6D (3D positions plus 3D attitude angles) solution to Simultaneous Localization and Mapping (SLAM) using a range and bearing sensor. It will be not assumed here that the sensor coordinate reference system coincide with that of the robot, which severely complicates the equations required for the implementation but in turn allows the usage of any number of sensors on a robot without restrictions. It is also briefly discussed two implementations in C++ within the context of the Mobile Robot Programming Toolkit (MRPT) project, corresponding to a straightforward $\mathcal{O}(N^3)$ and a more efficient $\mathcal{O}(N^2)$ algorithm. Update rates in excess of 120Hz are demonstrated for a simulated sensor.

An online version of this document is available in:
http://babel.isa.uma.es/mrpt/index.php/6D-SLAM

Contents

1 Definitions **2**

2 EKF equations **4**

2.1 Prediction . . . 4

2.1.1 The Motion model . . . 4

2.1.2 Updating the state vector – $\hat{x}_{k|k-1}$ . . . 5

2.1.3 Updating the covariance – $P_{k|k-1}$ . . . 5

2.1.4 Jacobians . . . 6

2.2 Update . . . 7

2.2.1 The Observation model . . . 7

2.2.2 Observation Noise – R . . . 8

2.2.3 Kalman innovation – S . . . 8

2.2.4 Calculation of the Jacobian $\frac{\partial h}{\partial x}$ . . . 9

3 Creating new landmarks **10**

3.1 Map expansion . . . 10

3.2 Inverse sensor model . . . 11

4 Complete Algorithms **12**

4.1 Naive EKF – $\mathcal{O}(N^3)$ . . . 12

4.2 A More Efficient Method – $\mathcal{O}(N^2)$ . . . 12

5 C++ implementation **14**

6 Results **15**

7 Comments **17**

A Homogeneous coordinates **18**

B Jacobians of the observation model **19**

Chapter 1

Definitions

Our goal is to estimate the 6D pose of a robot equipped with any kind of sensor capable of detecting 3D landmarks in its environment. The solution aims for real-time localization while simultaneously building a map of the environment.

Our implementation will make use of 3D Cartesian coordinates and the three angles yaw, pitch, and roll for attitudes. Following the notation of Davison *et al.* in [2], let $\mathbf{x_v}$ denote the vector of the vehicle pose:

$$\begin{aligned} \mathbf{x_v} &= [x\ y\ z\ \phi\ \chi\ \psi]^T \\ \phi &: \quad yaw, \ \ \chi: \ \ pitch, \ \ \psi: \ \ roll \end{aligned} \tag{1.1}$$

The geometry of such a coordinate configuration is illustrated in Fig. 1. At any instant of time k we will have a number L of landmarks or feature points in the map, each one described by its 3D global coordinates $\mathbf{y_i}$:

$$\mathbf{y_i} = [x_i\ y_i\ z_i]^T \tag{1.2}$$

The complete state vector of our problem is the concatenation of the vehicle pose and the position of all the landmarks, and is denoted with $\mathbf{x}$:

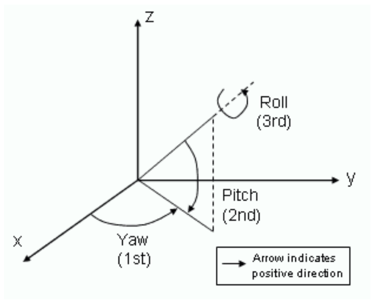

Figure 1.1: The coordinate system used in this work. Typically, yaw=0 (i.e. the +x axis) points in the robot forward direction.

$$\mathbf{x} = \begin{bmatrix} \mathbf{x_v} \\ \mathbf{y_1} \\ \mathbf{y_2} \\ \cdots \\ \mathbf{y_L} \end{bmatrix} \tag{1.3}$$

Due to uncertainties inherent to sensors and actuators, we must use a probabilistic representation of this state rather than keeping just one single estimation at each time step. As common in the SLAM literature, we model this joint distribution for each time step k as a multivariate Gaussian with mean $\hat{x}_{kk}$ and covariance matrix P_{kk} of size $(6+3L) \times (6+3L)$. It is advantageous for further reference to denote each of the submatrices of P_{kk} according to the involved variables:

$$P_{kk_{|6+3L\times 6+3L}} = \begin{pmatrix} P_{xx_{|6\times 6}} & P_{xy1_{|6\times 3}} & \cdots & P_{xyL_{|6\times 3}} \\ P_{y1x_{|3\times 6}} & P_{y1y1_{|3\times 3}} & \cdots & P_{y1yL_{|3\times 3}} \\ \cdots & \cdots & \cdots & \cdots \\ P_{yLx_{|3\times 6}} & P_{yLy1_{|3\times 3}} & \cdots & P_{yLyL_{|3\times 3}} \end{pmatrix} \tag{1.4}$$

It is common to initialize the vehicle pose to all zeros (i.e. the world reference is the robot starting position), while the covariance matrix should be initialized to zero, since this uncertainty will determinate the best the robot can localize itself from now on [3].

The estimated probability distribution evolves according to an Extended Kalman Filter (EKF) [4], a Bayesian filter that updates our current belief according to the new observations as they are gathered by the robot in successive time steps. The EKF consists on two steps: prediction and update, whose generic equations are given by:

$$\begin{aligned}
&\mathit{Prediction}: \\
\hat{x}_{k|k-1} &= f(\hat{x}_{k-1|k-1}, u_k) && (1.5) \\
P_{k|k-1} &= \frac{\partial f}{\partial x} P_{k-1|k-1} \frac{\partial f}{\partial x}^T + Q_k && (1.6) \\
&\mathit{Update}: \\
\tilde{y}_k &= z_k - h(\hat{x}_{k|k-1}) && (1.7) \\
S_k &= \frac{\partial h}{\partial x} P_{k|k-1} \frac{\partial h}{\partial x}^T + R_k && (1.8) \\
K_k &= P_{k|k-1} \frac{\partial h}{\partial x}^T S_k^{-1} && (1.9) \\
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k \tilde{y}_k && (1.10) \\
P_{k|k} &= (1 - K_k \frac{\partial h}{\partial x}) P_{k|k-1} && (1.11)
\end{aligned}$$

Where $f(\cdot)$ and $h(\cdot)$ stand for the transition and observation models of the system, respectively. The rest of the report is devoted to the calculation of each of the required terms in these equations, as well as to exploit their special configuration to speed up their implementation.

Chapter 2

EKF equations

In this chapter we derive all the expressions stated by the generic EKF equations (1.5)-(1.11). In Chapter 4 we will give more details about how to employ all these equations in the EKF implementation.

2.1 Prediction

We define a robot action u_k at time step k as any "motor actuation" performed by the robot since the last time step $k-1$. A typical situation for ground vehicles is to take the increment of the robot odometry (increments in x, y, and the heading yaw) as the action.

We will assume here that actions u_k are 6D pose increments since the last time step $k-1$. In practice, these increments can be estimations from visual odometry, inertial sensors, or they must just consist of planar pose increments from wheels odometry. In fact, the algorithm will also works without any kind of "actions": one alternative is to set $u = 0$ with a large uncertainty U, i.e. the robot is close to its last position up to a degree of uncertainty.

2.1.1 The Motion model

Since we assume static landmarks, the only variables of the state vector that actually change over time are those of the vehicle pose. At each time step, the pose of the vehicle x_v changes according to the transition model:

$$x_{v\{k\}} = f_v(x_{v\{k-1\}}, u_k) = x_{v\{k-1\}} \oplus u_k \tag{2.1}$$

Where $\oplus$ stands for the pose composition operator [5], and the components of u are $u = \{x_u \ y_u \ z_u \ \phi_u \ \chi_u \ \psi_u\}$. Using homogeneous coordinates it can be shown that the function f_v becomes:

$$\begin{cases} x_k = x_{k-1} + R_{11}x_u + R_{12}y_u + R_{13}z_u \\ y_k = y_{k-1} + R_{21}x_u + R_{22}y_u + R_{23}z_u \\ z_k = z_{k-1} + R_{31}x_u + R_{32}y_u + R_{33}z_u \\ \phi_k = \phi_{k-1} + \phi_u \\ \chi_k = \chi_{k-1} + \chi_u \\ \psi_k = \psi_{k-1} + \psi_u \end{cases} \tag{2.2}$$

Where R_{ij} stands for the corresponding entries in the 4×4 homogeneous coordinates matrix of x_{k-1} (see Appendix A).

2.1.2 Updating the state vector – $\hat{x}_{k|k-1}$

Thus, updating the state vector (1.3) according to the first step in the EKF (1.5) becomes simply modifying the first 6 elements of $\hat{x}$, that is, $\hat{x}_v$, as:

$$\hat{x}_{v\{k|k-1\}} = f_v(\hat{x}_{v\{k-1|k-1\}}, u) \tag{2.3}$$

Using (2.2), whereas all the landmark positions remain unmodified.

2.1.3 Updating the covariance – $P_{k|k-1}$

The second step in the EKF implies updating the whole covariance $P_{k|k-1}$ according to the new predicted vehicle pose $\hat{x}_{k|k-1}$. For this we need the Jacobian of the transition function relative to the state vector:

$$\left.\frac{\partial f}{\partial x}\right|_{6+3L\times 6+3L} = \begin{pmatrix} \left.\frac{\partial f_v}{\partial x_v}\right|_{6\times 6} & 0|_{6\times 3} & 0|_{6\times 3} & \cdots & 0|_{6\times 3} \\ 0|_{3\times 3} & I|_{3\times 3} & 0|_{3\times 3} & \cdots & 0|_{3\times 3} \\ 0|_{3\times 3} & 0|_{3\times 3} & I|_{3\times 3} & \cdots & 0|_{3\times 3} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0|_{3\times 3} & 0|_{3\times 3} & 0|_{3\times 3} & \cdots & I|_{3\times 3} \end{pmatrix} \tag{2.4}$$

If we expand (1.6) taking into account the specific structure of the above Jacobian, we have:

$$\begin{aligned} P_{k|k-1} &= \frac{\partial f}{\partial x} P_{k-1|k-1} \frac{\partial f}{\partial x}^T + Q_k \qquad (2.5) \\ &= \left(\begin{array}{c|ccc} \frac{\partial f_v}{\partial x_v} P_{xx} \frac{\partial f_v}{\partial x_v}^T & \frac{\partial f_v}{\partial x_v} P_{xy1} & \cdots & \frac{\partial f_v}{\partial x_v} P_{xyL} \\ \hline P_{y1x} \frac{\partial f_v}{\partial x_v}^T & & & \\ \cdots & & Unmodified & \\ P_{yLx} \frac{\partial f_v}{\partial x_v}^T & & & \end{array}\right) + Q_k \end{aligned}$$

Note how just the first row and column are modified in this step (actually, the first column is the transpose of the first row, so it is enough to make half the computations).

The Q_k term stands for the uncertainty in the new vehicle state due to the uncertainty in the robot action (e.g. noisy odometry). Recall that we denote the increment in pose as the 6-length vector u, which is associated a 6×6 covariance matrix U. Thus, Q_k is computed through the Jacobian of the transition function with respect to the robot action:

$$Q_k|_{6+3L\times 6+3L} = \left.\frac{\partial f}{\partial u}\right|_{6+3L\times 6+3L} U \frac{\partial f}{\partial u}^T \tag{2.6}$$

Since this Jacobian is zero for the landmark variables we can just compute the 6×6 covariance Q_{k_v}:

$$Q_{k_v}|_{6\times 6} = \left.\frac{\partial f_v}{\partial u}\right|_{6\times 6} U_{|6\times 6} \frac{\partial f_v}{\partial u}^T \tag{2.7}$$

and add it to the top-left sub-matrix (the updated vehicle covariance) in (2.5). Note that there is no reason to assume here that U is a diagonal matrix since it would imply a negligible speed-up while a full-matrix can model more accurately the possible correlations in the incremental pose estimations.

2.1.4 Jacobians

Here we derive the expressions for the Jacobians $\frac{\partial f_v}{\partial x_v}$ and $\frac{\partial f_v}{\partial u}$ which have been used in the previous section.

Calculation of $\frac{\partial f_v}{\partial x_v}$

The elements of this 6×6 Jacobian for our case of 6D odometry can be obtained from (2.2) and are:

$$\frac{\partial f_v}{\partial x_v} = \begin{pmatrix} 1 & 0 & 0 & F_{14} & F_{15} & F_{16} \\ 0 & 1 & 0 & F_{24} & F_{25} & F_{26} \\ 0 & 0 & 1 & 0 & F_{35} & F_{36} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

where:
F_{14} = -sy*cp*xu+(-sy*sp*sr-cy*cr)*yu+(-sy*sp*cr+cy*sr)*zu
F_{15} = -cy*sp*xu + cy*cp*sr*yu+cy*cp*cr*zu
F_{16} = (cy*sp*cr+sy*sr)*yu+(-cy*sp*sr+sy*cr)*zu
F_{24} = cy*cp*xu+(cy*sp*sr-sy*cr)*yu+(cy*sp*cr+sy*sr)*zu
F_{25} = -sy*sp*xu+sy*cp*sr*yu+sy*cp*cr*zu
F_{26} = (sy*sp*cr-cy*sr)*yu+(-sy*sp*sr-cy*cr)*zu
F_{35} = -cp*xu-sp*sr*yu-sp*cr*zu
F_{36} = cp*cr*yu-cp*sr*zu

Here $(xu\ yu\ zu)^T$ is the 3D position increment of the action u (e.g. visual odometry), and the following abbreviations have been used for quantities related to the vehicle attitude angles at time step $k-1$:

$$\begin{aligned} sy &= \sin\phi_{k-1} \quad cy = \cos\phi_{k-1} \quad (Yaw) \\ sp &= \sin\chi_{k-1} \quad cp = \cos\chi_{k-1} \quad (Pitch) \\ sr &= \sin\psi_{k-1} \quad cr = \cos\psi_{k-1} \quad (Roll) \end{aligned}$$

Calculation of $\frac{\partial f_v}{\partial u}$

It is straightforward from (2.2) to arrive at:

$$\frac{\partial f_v}{\partial u} = \left(\begin{array}{ccc|c} R_{11} & R_{12} & R_{13} & \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & \\ \hline & 0 & & I_3 \end{array}\right) \tag{2.9}$$

where the R_{ij} components are the corresponding elements of the homogeneous matrix for the robot pose $\hat{x}_{k-1|k-1}$ (see Appendix A).

2.2 Update

2.2.1 The Observation model

Given the robot pose $\mathbf{x_v}$ and the 3D coordinates of a landmark $\mathbf{y_i} = (x_i \; y_i \; z_i)^T$, our observation model for a bearing-range sensor is:

$$z_i = h_i(\mathbf{x_v}, \mathbf{y_i}) \tag{2.10}$$

where each observation z_k comprises the following three values:

$$z_k = \begin{pmatrix} r \\ \alpha \\ \beta \end{pmatrix} \quad \rightarrow \begin{cases} r : \text{Range (distance) to the landmark} \\ \alpha : \text{Yaw (azimuth) to the landmark} \\ \beta : \text{Pitch (elevation) to the landmark} \end{cases} \tag{2.11}$$

These values are computed through:

$$r = \sqrt{\bar{x}_i^2 + \bar{y}_i^2 + \bar{z}_i^2} \tag{2.12}$$

$$\alpha = \arctan \frac{\bar{y}_i}{\bar{x}_i} \tag{2.13}$$

$$\beta = -\arctan \frac{\bar{z}_i}{\sqrt{\bar{x}_i^2 + \bar{y}_i^2}} \tag{2.14}$$

where $\mathbf{\bar{y}_i} = (\bar{x}_i \; \bar{y}_i \; \bar{z}_i)^T$ is the landmark y_i in coordinates relative to the vehicle $\mathbf{x_v}$. We also want to incorporate in the calculations the pose of the sensor relative to the robot, which we will denote as $\mathbf{x_s}$.

Then, the relative coordinates can be computed using homogeneous coordinates, in such a way that:

$$\mathbf{\bar{y}_i} = \mathbf{y_i} \ominus (\mathbf{x_v} \oplus \mathbf{x_s}) \tag{2.15}$$

becomes:

$$\left(\begin{array}{ccc|c} & & & \bar{x}_i \\ & & & \bar{y}_i \\ & & & \bar{z}_i \\ \hline 0 & 0 & 0 & 1 \end{array}\right) = (R(\mathbf{x_v})R(\mathbf{x_s}))^{-1} \left(\begin{array}{ccc|c} & & & x_i \\ & 0 & & y_i \\ & & & z_i \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{2.16}$$

Note that only the three marked components from the resulting matrix are required for the function h_i. Using the MATLAB symbolic toolbox the inverse of the product of the rotation matrices has been computed analytically, so the required Jacobians can be computed (see sections 2.2.4).

2.2.2 Observation Noise – R

The matrix R in (1.8) stands for the uncertainty due to the sensor noise, and is typically modelled trough a diagonal matrix:

$$R = \begin{pmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\alpha^2 & 0 \\ 0 & 0 & \sigma_\beta^2 \end{pmatrix} \tag{2.17}$$

where each value is the variance of the corresponding range or bearing measurement. These are parameters of the system, and must be established from a characterization of the sensor or heuristically.

The fact of this matrix being diagonal, which implies the reasonable assumption of uncorrelated errors in each of the measurement components and between landmarks, is the basis for the optimized EKF implementation presented later on [2].

2.2.3 Kalman innovation – S

The full matrix S

In general, the matrix S is computed by 1.8 as:

$$S_k = \frac{\partial h}{\partial x} P_{k|k-1} \frac{\partial h}{\partial x}^T + R_k \tag{2.18}$$

Note how S_k becomes a fully populated $6+3L \times 6+3L$ matrix, in despite of the peculiar structure of the Jacobian $\frac{\partial h}{\partial x}$ (see 2.21).

The scalar S_i^j

In turn, the innovation matrix becomes a scalar if the components of the observations are considered sequentially once at a time (range, yaw, pitch, range, yaw, pitch, etc.) as remarked in [2].

Let S_i^j be this scalar for the j'th component of the i'th observed landmark. Then:

$$\begin{aligned} S_i^j &= \frac{\partial h_i^j}{\partial x} \begin{pmatrix} P_{xx|6\times 6} & P_{xyi|6\times 3} \\ P_{yix|3\times 6} & P_{yiyi|3\times 3} \end{pmatrix} \frac{\partial h_i^j}{\partial x}^T + R_{jj} \\ &= \frac{\partial h_i^j}{\partial x_v} P_{xx} \frac{\partial h_i^j}{\partial x_v}^T + \frac{\partial h_i^j}{\partial y_i} P_{yix} \frac{\partial h_i^j}{\partial x_i}^T + \frac{\partial h_i^j}{\partial x_v} P_{xyi} \frac{\partial h_i^j}{\partial y_i}^T + \frac{\partial h_i^j}{\partial y_i} P_{yiyi} \frac{\partial h_i^j}{\partial y_i}^T + R_{jj} \\ &= \frac{\partial h_i^j}{\partial x_v} P_{xx} \frac{\partial h_i^j}{\partial x_v}^T + 2\frac{\partial h_i^j}{\partial y_i} P_{yix} \frac{\partial h_i^j}{\partial x_i}^T + \frac{\partial h_i^j}{\partial y_i} P_{yiyi} \frac{\partial h_i^j}{\partial y_i}^T + R_{jj} \end{aligned} \tag{2.19}$$

where the partial Jacobians are the j'th row of $\frac{\partial h_i^j}{\partial x}$ and R_{jj} is the j'th diagonal element of (2.17). Note how this R term assure that S_i^j will be always invertible. Note as well that the last step in the equation above (the union of two terms) is possible just in the case of S_i^j being scalar and does not hold in general.

2.2.4 Calculation of the Jacobian $\frac{\partial h}{\partial x}$

In general, we have that this Jacobian of the full vector of observations $h = \{h_1, h_2, ...\}$ is composed of:

$$\left.\frac{\partial h}{\partial x}\right|_{3L\times(6+3L)} = \begin{pmatrix} \frac{\partial h_1}{\partial x_v} & \frac{\partial h_1}{\partial y_1} & \frac{\partial h_1}{\partial y_2} & ... \\ \frac{\partial h_2}{\partial x_v} & \frac{\partial h_2}{\partial y_1} & \frac{\partial h_2}{\partial y_2} & ... \\ \frac{\partial h_3}{\partial x_v} & \frac{\partial h_3}{\partial y_1} & \frac{\partial h_3}{\partial y_2} & ... \\ ... & ... & ... & ... \end{pmatrix} \tag{2.20}$$

But due to the fact that the observation of each landmark i depends on the vehicle pose and its corresponding landmark y_i, the matrix is actually sparsely populated:

$$\left.\frac{\partial h}{\partial x}\right|_{3L\times(6+3L)} = \begin{pmatrix} \frac{\partial h_1}{\partial x_v} & \frac{\partial h_1}{\partial y_1} & 0 & ... \\ \frac{\partial h_2}{\partial x_v} & 0 & \frac{\partial h_2}{\partial y_2} & ... \\ \frac{\partial h_3}{\partial x_v} & 0 & 0 & ... \\ ... & ... & ... & ... \end{pmatrix} \tag{2.21}$$

Hence we only need to compute the two partial Jacobians $\frac{\partial h_i}{\partial x_v}$ and $\frac{\partial h_i}{\partial y_i}$ for each landmark y_i:

$$\left.\frac{\partial h_i}{\partial x}\right|_{3\times 6+3L} = \left(\left.\frac{\partial h_i}{\partial x_v}\right|_{3\times 6} \; ... \; \left.\frac{\partial h_i}{\partial y_i}\right|_{3\times 3} \; ... \right) \tag{2.22}$$

where each row

$$\left.\frac{\partial h_i^j}{\partial x}\right|_{1\times 6+3L} = \left(\left.\frac{\partial h_i^j}{\partial x_v}\right|_{1\times 6} \; ... \; \left.\frac{\partial h_i^j}{\partial y_i}\right|_{1\times 3} \; ... \right) \tag{2.23}$$

for $j = 1, 2, 3$ corresponds to each of the three components of a single range-bearing observation. The procedure to compute the partial Jacobians is given in the Appendix B.

Chapter 3

Creating new landmarks

A common step to both algorithms discussed in Chapter 4 is the introduction of new landmarks the first time they are observed, what is performed as follows.

3.1 Map expansion

Let $L-1$ denote the number of landmarks in the map at some time step. Then, introducing a new landmark L requires the evaluation of the inverse sensor model $y_L(x_v, h_L)$, with x_v being the vehicle pose and h_L the range and bearing observation of the landmark. The system state is expanded with the new landmark estimation:

$$\begin{aligned} \hat{x} &\leftarrow (\hat{x} \; \hat{y}_L) && (3.1) \\ \hat{y}_L &= y_L(\hat{x}_v, h_L) && (3.2) \end{aligned}$$

and the covariance matrix is also expanded with a new row and column, such as [2]:

$$P_{kk} = \begin{pmatrix} P_{xx} & P_{xy_1} & \dots & P_{xy_{L-1}} & P_{xx} \frac{\partial y_L}{\partial x_v}^T \\ P_{y_1 x} & P_{y_1 y_1} & \dots & P_{y_1 y_{L-1}} & P_{y_1 x} \frac{\partial y_L}{\partial x_v}^T \\ \dots & \dots & \dots & \dots & \dots \\ P_{y_{L-1} x} & P_{y_{L-1} y_1} & \dots & P_{y_{L-1} y_{L-1}} & P_{y_{L-1} x} \frac{\partial y_L}{\partial x_v}^T \\ \frac{\partial y_L}{\partial x_v} P_{xx} & \frac{\partial y_L}{\partial x_v} P_{xy_1} & \dots & \frac{\partial y_L}{\partial x_v} P_{xy_{L-1}} & A \end{pmatrix} \quad (3.3)$$

where:

$$A = \frac{\partial y_L}{\partial x_v} P_{xx} \frac{\partial y_L}{\partial x_v}^T + \frac{\partial y_L}{\partial h_L} R \frac{\partial y_L}{\partial h_L}^T \quad (3.4)$$

with R being the sensor noise covariance matrix given in (2.17).

3.2 Inverse sensor model

The inverse model $y_L(x_v, h)$ simply consists of a projection of the landmark from the sensor point using the range (h_r) and bearing angles (yaw h_α and pitch h_β), that is:

$$y_L(x_v, h_L) = \left(\begin{array}{c} x_L \\ y_L \\ z_L \end{array} \right) = x_v \oplus x_s \oplus \left(\begin{array}{c} h_r \cos h_\alpha \cos h_\beta \\ h_r \sin h_\alpha \cos h_\beta \\ -h_r \sin h_\beta \end{array} \right) \tag{3.5}$$

with x_s being the relative pose of the sensor on the robot. The corresponding Jacobians $\frac{\partial y_L}{\partial x_v}$ and $\frac{\partial y_L}{\partial h_L}$ have been computed using the MATLAB symbolic toolbox.

Chapter 4

Complete Algorithms

In this chapter we summarize two implementations of an EKF for range-bearing SLAM.

4.1 Naive EKF – $\mathcal{O}(N^3)$

A straightforward implementation consists of following all the steps (1.5)-(1.11) taking the whole vector of observations z_k at once. The algorithm then is:

1. Predict new robot pose $\hat{x_v}_{k\|k-1}$ using (2.2).	$\mathcal{O}(1)$
2. Modify the covariance following (2.5).	$\mathcal{O}(N)$
3. Predict observations h_i (2.12) and Jacobians of h_i.	$\mathcal{O}(N)$
4. Compute the whole matrix S (2.18).	$\mathcal{O}(N^3)$
5. Inverse of S and computation of K_k (1.9).	$\mathcal{O}(N^3)$
6. Update the filter state vector $\hat{x_v}_{k\|k}$ using (1.10).	$\mathcal{O}(N^2)$
6. Update the filter covariance $P_{k\|k}$ using (1.11).	$\mathcal{O}(N^3)$
8. If necessary, introduce new landmarks in the map.	$\mathcal{O}(N)$

Where N stands here for the number of landmarks, the most decisive quantity for complexity analysis since the rest of dimensions remain constant as the filter evolves. This approach is therefore unpractical for real-time purposes for maps of more than a few dozens of landmarks.

4.2 A More Efficient Method – $\mathcal{O}(N^2)$

This alternative method consists of considering sequentially the observations, taking one scalar value at each time. It has been already shown in (2.19) that in this case the Kalman innovation matrix S_i^j becomes a scalar.

Now we derive the expressions for the column matrix Kalman gain K_i^j, for the j'th scalar component of the i'th observed landmark:

$$\begin{aligned} K_i^j|_{6+3L\times 1} &= P_{k|k-1}\frac{\partial h_i^j}{\partial x}^T \frac{1}{S_i^j} \\ &= \left(\begin{pmatrix} P_{xx} \\ P_{xy_1} \\ P_{xy_2} \\ \dots \end{pmatrix} \frac{\partial h_i^j}{\partial x_v}^T + \begin{pmatrix} P_{xy_i} \\ P_{y_1 y_i} \\ P_{y_2 y_i} \\ \dots \end{pmatrix} \frac{\partial h_i^j}{\partial y_i}^T \right) \frac{1}{S_i^j} \end{aligned} \tag{4.1}$$

Note how the sparse structure of the Jacobian allows the separation of this equation in two parts where only two columns of the full $P_{k|k-1}$ are involved.

Once computed this column Kalman gain, the position of the involved landmark and the vehicle is also simplified:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_i^j(z_i^j - h_i^j) \tag{4.2}$$

where the scalar $(z_i^j - h_i^j)$ represents the difference between the actual measurement and the prediction of the filter.

Regarding the update of the covariance matrix P_{kk}, following [2]:

$$P_{k|k} = P_{k|k-1} - S_i^j K_i^j K_i^{jT} \tag{4.3}$$

The multiplication $K_i^j K_i^{jT}$ can be made faster by noting that its result must be symmetric thus it is only required to compute half the values.

Finally, the overall algorithm can be summarized as:

1. Predict new robot pose $\hat{x_v}_{k\|k-1}$ using (2.2).	$\mathcal{O}(1)$
2. Modify the covariance following (2.5).	$\mathcal{O}(N)$
3. For each of the M observed landmarks i:	$\times\mathcal{O}(M)$
3.1. Compute h_i (2.12) and its Jacobian.	$\mathcal{O}(1)$
3.2. For each dimension j of the observation h_i:	
3.2.1 Compute the scalar S_i^j (2.19).	$\mathcal{O}(1)$
3.2.2 Computation of K_i^j (4.1).	$\mathcal{O}(N)$
3.2.3 Update the filter state vector $\hat{x_v}_{k\|k}$ using (4.2).	$\mathcal{O}(N)$
3.2.4 Update the filter covariance $P_{k\|k}$ using (4.3).	$\mathcal{O}(N^2)$
4. If necessary, introduce new landmarks in the map.	$\mathcal{O}(N)$

Although there are two nested loops, they must be executed a fixed number of times that depends mainly on the number of observed landmarks M, thus the complexity of the algorithm becomes $\mathcal{O}(M \cdot N^2)$. However, in practice M remains bounded, hence this algorithm reduces the complexity from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^2)$.

Chapter 5

C++ implementation

Both the naive and the improved EKF algorithms have been implemented in the MRPT C++ library, concretely in the class `MRML::CRangeBearingKFSLAM`. Internally, this class calls the generic EKF implemented in `UTILS::CKalmanFilterCapable` which can be employed by any problem for arbitrary sizes the the state, observations, and landmark vectors. For more information refer to [1].

Chapter 6

Results

A simulated dataset has been generated for a square robot path moving on a plane while detecting 3D landmarks. The experiment includes a loop closure, whose effects can be observed in Fig. 6. At the end of the experiment, 92 landmarks are mapped out of a total of 100. This is due to the limited range and field of view of the simulated sensor. In average 6.6 landmarks are observed simultaneously over the 2199 time steps of the experiment.

Statistical results are summarized next for the execution of each of the algorithms, including the errors in the estimated landmark 3D positions:

Value	**Naive method**	**Improved method**
Overall time	293 sec	17.15 sec
Average execution rate	7.5 Hz	128.2 Hz
Average landmark error	0.157 m	0.138 m
Maximum landmark error	0.278 m	0.245 m
Minimum landmark error	0.037 m	0.064 m

This results are for an Intel Core2 Duo T7500 @ 2.20GHz. The reduced average error in the case of the improved algorithm may be caused by the more exact computation of the Jacobians, since they are linearized at increasingly more accurate positions as the scalar components of the observation vector are integrated sequentially.

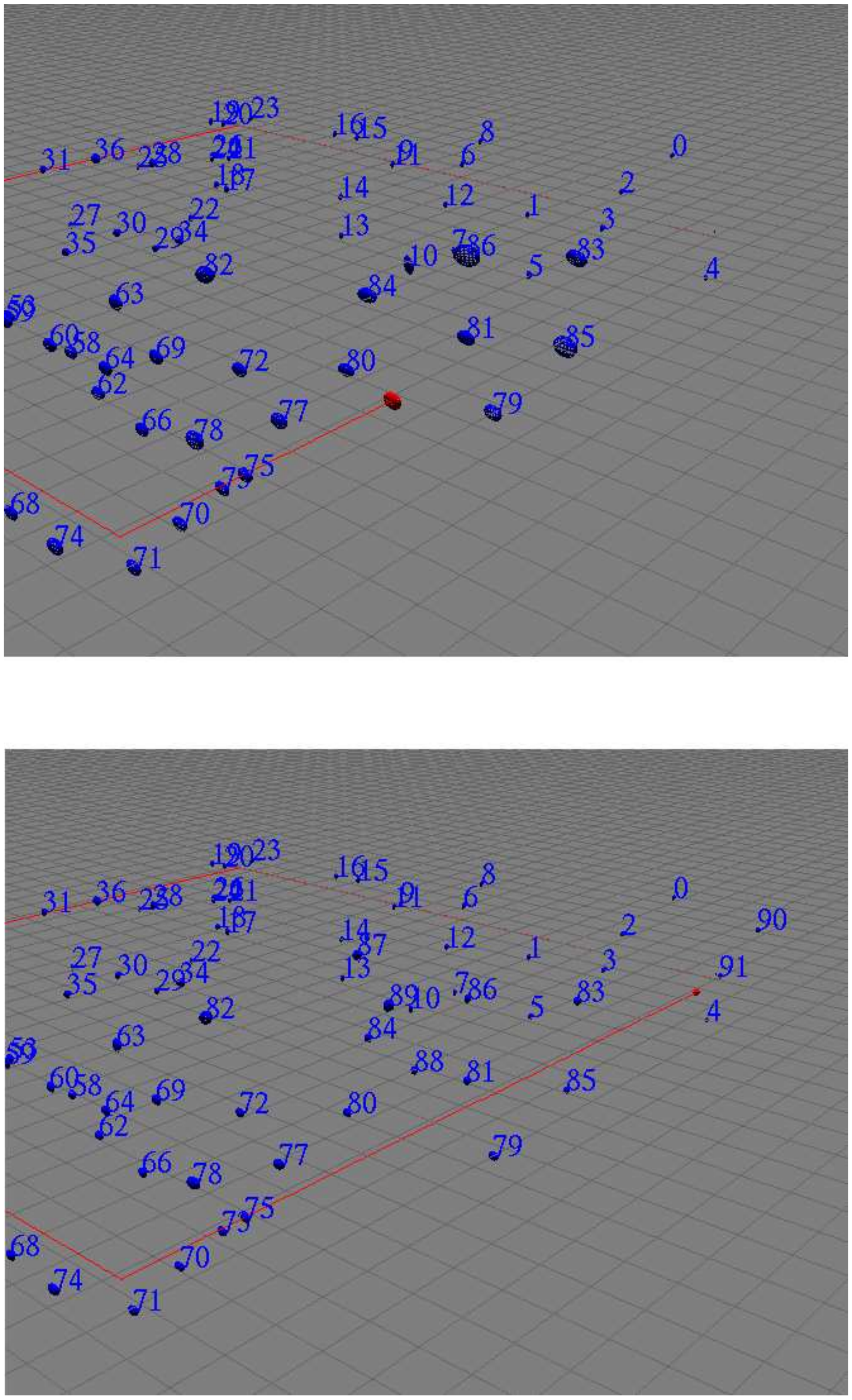

Figure 6.1: Two snapshots of the filter state, before (top) and after (bottom) closing a loop.

Chapter 7

Comments

This report has given a detailed description of an efficient implementation of an EKF for the specific problem of 6D SLAM. It has been shown the efficiency of the method for small maps, which immediately raises potential applications as:

- Stereo Vision-based SLAM: Matched features from a stereo camera are the perfect input to the algorithm in the form of range and direction angles.
- Visual Odometry: Used incrementally, i.e. taking the map as the last observation, the EKF presents a grounded method to estimate the displacement and associated covariance of full 6D pose increments between consecutive time steps.
- Image tracking: The generic implementation allows the easy development of efficient trackers in the space of the image plane.

Appendix A

Homogeneous coordinates

Let B be a change of coordinates in 3D described through a increment in x, y, and z and rotations in yaw, pitch, and roll, that is,

$$B = \{x\ y\ z\ \phi\ \chi\ \psi\},$$

The homogeneous coordinates matrix of B is given by:

$$\begin{aligned} R(B) &= \begin{pmatrix} R_{11} & R_{12} & R_{13} & x \\ R_{21} & R_{22} & R_{23} & y \\ R_{31} & R_{32} & R_{33} & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} \cos\phi\cos\chi & \cos\phi\sin\chi\sin\psi - \sin\phi\cos\psi & \cos\phi\sin\chi\cos\psi + \sin\phi\sin\psi & x \\ \sin\phi\cos\chi & \sin\phi\sin\chi\sin\psi + \cos\phi\cos\psi & \sin\phi\sin\chi\cos\psi - \cos\phi\sin\psi & y \\ -\sin\chi & \cos\chi\sin\psi & \cos\chi\cos\psi & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{A.1}$$

Composing two poses A and B is equivalent to multiplying their corresponding homogeneous matrices. For example, to obtain $C = A \oplus B$ we can compute $R(A)R(B)$, what gives us the homogeneous matrix of C.

The equivalent operation in the MRPT C++ library is denoted by the + operator between `MRML::CPose3D` objects, e.g.

```
CPose3D A(1,2,3),B(4,0,0),C;
C = A + B;
```

Appendix B

Jacobians of the observation model

The following abbreviations are used in this appendix:

$$\begin{aligned}
x0 &= \hat{x}_v \\
y0 &= \hat{y}_v \\
z0 &= \hat{z}_v \\
y &= \hat{\phi}_v \\
p &= \hat{\chi}_v \\
r &= \hat{\psi}_v \\
x0s &= x_s \\
y0s &= y_s \\
z0s &= z_s \\
ys &= \phi_s \\
ps &= \chi_s \\
rs &= \psi_s
\end{aligned}$$

The complexity of the obtained equations is a consequence of allowing an arbitrary pose $\mathbf{x}_s = (x_s\ y_s\ z_s\ \phi_s\ \chi_s\ \psi_s)^T$ for each sensor on the robot. The complete formulas can be found in the source code of the MRPT C++ library, concretely in the file `MRML/CRangeBearingKFSLAM.cpp`.

We reproduce next the MATLAB scripts employed to derive the corresponding symbolic expressions. Here the 3×6 Jacobian $\frac{\partial h_i}{\partial x_v}$ is denoted by `J_hi_xv` and the 3×3 Jacobian $\frac{\partial h_i}{\partial y_i}$ is denoted by `J_hi_yi`:

```
% compute_jacobians.m script
syms x0 y0 z0 y p r         % robot 6D pose
syms x0s y0s z0s ys ps rs   % sensor 6D pose on robot
syms xi yi zi Xi     % landmakr absolute 3D point
syms xi_ yi_ zi_     % landmark relative 3D point

% Homog. matrix for the robot pose:
R_x = [ cos(y)*cos(p) cos(y)*sin(p)*sin(r)-sin(y)*cos(r) cos(y)*sin(p)*cos(r)+sin(y)*sin(r) x0;
        sin(y)*cos(p) sin(y)*sin(p)*sin(r)+cos(y)*cos(r) sin(y)*sin(p)*cos(r)-cos(y)*sin(r) y0;
       -sin(p) cos(p)*sin(r) cos(p)*cos(r) z0;
        0   0   0 1];

% Homog. matrix for the sensor pose on robot:
R_xs = [ cos(ys)*cos(ps) cos(ys)*sin(ps)*sin(rs)-sin(ys)*cos(rs) cos(ys)*sin(ps)*cos(rs)+sin(ys)*sin(rs) x0s;
        sin(ys)*cos(ps) sin(ys)*sin(ps)*sin(rs)+cos(ys)*cos(rs) sin(ys)*sin(ps)*cos(rs)-cos(ys)*sin(rs) y0s;
       -sin(ps) cos(ps)*sin(rs) cos(ps)*cos(rs) z0s;
        0   0   0 1];

% Inverse of the composition: x (+) x_s
R_xxs = R_x * R_xs;

disp('Computing inverse matrix....');
[R_xxs_1, how_R]=simple(inv(R_xxs));

Xi=[ 1 0 0 xi;
     0 1 0 yi;
     0 0 1 zi;
     0 0 0 1 ];

RES = R_xxs_1*Xi;

xi_ = simple(RES(1,4));
yi_ = simple(RES(2,4));
zi_ = simple(RES(3,4));

xi_
yi_
zi_

% The observation model:
syms H h_range h_yaw h_pitch  J_hi_xv J_hi_yi

h_range = sqrt(xi_^2+yi_^2+zi_^2);

h_yaw   = atan(yi_/xi_);

h_pitch = -atan(zi_/ sqrt( xi_^2 + yi_^2 ) );

H=[ h_range ; h_yaw ; h_pitch ];


disp('Computing jacobian wrt xv....');

J_hi_xv=jacobian(H,[x0 y0 z0 y p r]);


disp('Computing jacobian wrt yi....');

J_hi_yi=jacobian(H,[xi yi zi]);
```

Bibliography

[1] J.L. Blanco. The Mobile Robot Programming Toolkit (MRPT) website, 2008.

[2] A.J. Davison, I. Reid, N. Molton, and O. Stasse. MonoSLAM: Real-Time Single Camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6), June 2007.

[3] M. Dissanayake, P. Newman, S. Clark, HF Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.

[4] S.J. Julier and J.K. Uhlmann. A new extension of the Kalman filter to nonlinear systems. *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*, 3, 1997.

[5] R. Smith, M. Self, and P. Cheeseman. A stochastic map for uncertain spatial relationships. *The fourth international symposium on Robotics Research*, pages 467–474, 1988.