MICHELE MAROSTICA

A LOOP CLOSURE APPROACH FOR HUMANOIDS NAVIGATION

A LOOP CLOSURE APPROACH FOR HUMANOIDS NAVIGATION

Candidate: MICHELE MAROSTICA

Advisor: PROF. ENRICO PAGELLO
Co-Advisor: PH.D. ALBERTO PRETTO

An approach to the loop closure problem, robust to severe view-point changes.

Laurea Magistrale in Ingegneria Informatica
Department of Information Engineering
Università degli Studi di Padova

October 2013

Michele Marostica: *A loop closure approach for humanoids navigation*, An approach to the loop closure problem, robust to severe view-point changes., © October 2013

SUPERVISORS:
Prof. Enrico Pagello
Ph.D. Alberto Pretto

LOCATION:
Padova

Dedicated to my whole family, my friends and Silvia that supported me a lot in the work for this thesis.

A special thanks goes to my advisor: Prof. Enrico Pagello and my supervisor: Ph.D. Alberto Pretto for their time, willingness and for everything that they taught to me. I'd like to thank also Mauro Antonello and all the people: professors, researchers and students that I encountered in the IAS-Lab for their hospitality and courtesy.

ABSTRACT

In this thesis we propose a novel approach to the loop closure problem for humanoid robots navigation, an approach robust to view-point severe changes.

Vision-based state-of-the-art works on the loop closure problem works well if the robot re-pass through the same place with the same or a very similar view-point.

The proposed approach work under the assumption of planar features (derived from the Manhattan assumption). We estimate the planes of the feature's points and then extract image's patches that lies on such planes. On these patches, that are an estimated frontal view of the features, we compute the descriptors that are used to perform the matches versus the map.

The planes extraction is done with the ego-motion estimation data that is provided with the input dataset.

With these view-point normalized features, we demonstrate the possibility to perform loop closure detection under severe view-point changes.

SOMMARIO

In questa tesi proponiamo un nuovo approccio al problema del "loop closure" per robot umanoidi, approccio che risulta robusto a importanti cambi del punto di vista.

I lavori attuali dello stato dell'arte, che vogliono risolvere il problema del loop closure e che si basano sulla visione, lavorano bene se il robot ripassa nello stesso posto con lo stesso o con un simile punto di vista.

L'approccio proposto lavora sotto l'assunzione di feature planari (una derivazione dell'assunzion Manhattan). Vengono stimati i piani su cui poggiano le feature. Poi vengono estratti degli intorni dell'immagine nei punti delle feature che stanno sul piano stimato. Questi intorni o patch sono una stima di una vista frontale delle feature, su essi vengono calcolati i descrittori che saranno poi utilizzati nei confronti contro la mappa.

L'estrazione dei piani è ottenuta grazie ai dati di stima dell'ego-motion forniti assieme al dataset usato.

Con queste feature normalizzate rispetto al punto di vista dimostriamo la possibilità di segnalare un loop closure anche con severi cambiamenti del punto di vista.

CONTENTS

LIST OF FIGURES

ACRONYMS

SLAM Simultaneous Localizationa and Mapping

PDF Probability Density Function

PF Particle Filter

IMU Inertial Measurement Unit

SAD Sum of Absolute Difference

SSD Sum of Squared Difference

NCC Normalized Cross Correlation

EKF Extended Kalman Filter

RBPF Rao-Blackwellized Particle Filter

ML Maximum Likelihood

KLD Kullback-Lieber Divergence

IDF Inverse Document Frequency

RANSAC Random Sample Consensus

NV Naive Vector

BOW Bag of Words

NNDR Nearest Neighbour Distance Ratio

PCL Point Cloud Library

LMFIT Levenberg-Marquardt least-squares minimization

1 INTRODUCTION

1.1 ORGANIZATION OF THE THESIS

The aim of this thesis is to explore some solution to the loop closure problem for humanoids navigation in indoor environment.

Such indoor environment are very challenging since in most structured buildings spaces look very similar, in some cases the appearance of different floor levels are equal.

Vision based approaches suffer a lot this problematic. If not enough state-of-the-art works demonstrated to work well when closing the loop with a very similar view-point. We demonstrate the possibility to obtain features descriptor more invariant to view-point changes by extracting an estimated frontal view of the features.

The first part of this thesis is an introduction to the Simultaneous Localizationa and Mapping (SLAM) problem of which the loop closure is a sub-problem. Followed by a description of some common humanoid robots.

The second part is an overview of the tools and algorithms included in the probabilistic framework adopted by the community to solve the SLAM problem.

In the third part we test a state-of-the-art algorithm, FAB-MAP, on an indoor environment and in the subsequent section we describe our approach and we compare the results.

1.2 THE SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM) PROBLEM

Thanks to the high level of degree of freedom that humanoid robots have, they can move into a three dimensional space in many ways. They can walk, they can crawl or they can go up and down the stairs. In order to achieve high level tasks a humanoid robot must know it's pose and the environment that surround it. This can be viewed as answering at two robot's characteristic questions: *Where i am?* and *What does the world look like?*

Answer to these questions, when as input you have only the set of measurements and the set of controls applied to the robot, means to solve the following problem: the SLAM problem. The SLAM problem is actually well defined in the robotics community as a moving sensor platform (wheeled, humanoids, etc.) constructing a *representation of its environment* while concurrently estimating its *ego-motion*.

In this document I will provide a description of some state-of-the-art approaches to the SLAM problem for humanoid robots with a focus on the loop closure problem.

Although some recent approaches reached good results in various large outdoor environment [13] and [4], the SLAM problem still remains quite open, and a robust solution is not ready yet.

Sometimes this is due to some strict assumptions that each system takes into account in its approximations. Most of time they assume independence between variables or they approximate the correlation structure.

Furthermore, state of the art works cannot reach a precise geometric consistency in large scale maps and in the best of my knowledge there has not been attempt to solve drastic failure recovery with humanoid robots.

As also stated in [20] persistence is still a problem since a robot should operate robustly for a long period of time. If it needs to do tasks that make it pass in the same environment different times, it should be able to reuse the map. But at the same time a robot cannot see its life as a *big one mission*, it would be infeasible for computational costs. As ideal the robot should be capable to expand the map when it sees new areas of the environment while improve the old parts when it repass throw them.

1.2.1 *Problem definition*

According to literature (see [1], [29]) the right framework for solving the *chicken-egg* SLAM problem is probabilistic. Inputs are a set of measurements $z_{1:t}$ and a set of controls $u_{1:t}$, the subscripts mean that they are measurements and controls from time 1 to t. There are two formulations of the problem: the first is the *online SLAM problem*, it involves estimating the posterior over the actual pose along with the map:

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{1}$$

Where x_t is the pose of the robot a time t and m is the map.

The second formulation is the *full SLAM problem* which has the goal to calculate a posterior over the entire path $x_{1:t}$ instead of just the current pose x_t:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{2}$$

The difference between on-line and full SLAM problems has consequences on the type of algorithms that can be used on. The on-line version can be viewed as the resulting of integrating out past poses from the full SLAM problem [29]:

$$p(x_t, m|z_{1:t}, u_{1:t}) = \int\int \cdots \int p(x_{1:t}, m|z_{1:t}, u_{1:t})\, dx_1\, dx_2 \ldots dx_{t-1} \quad (3)$$

Many algorithms for the on-line SLAM problem are incremental: they discard past measurements and controls once they have been processed, it means that above integrations are made one at a time.

A key aspect of the SLAM problem is that it involves a mixture of *continuous* and *discrete* estimation components: the location of objects in the map and the robot pose are continuous but the reasoning on correspondence between newly detected object and object previously detected is discrete.

Given that, inferring correspondences between newly detected object and object previously detected is fundamental to achieve good results it is preferable to explicitly put this variable into the problem definition:

$$p(x_t, m, c_t|z_{1:t}, u_{1:t}) \quad (4)$$

for the on-line case and

$$p(x_{1:t}, m, c_t|z_{1:t}, u_{1:t}) \quad (5)$$

for the full SLAM.

So now we get the on-line posterior from:

$$p(x_t, m, c_t|z_{1:t}, u_{1:t}) = \int\int \cdots \int \sum_{c_1}\sum_{c_2} \cdots \sum_{c_{t-1}} p(x_{1:t}, m, c_{1:t}|z_{1:t}, u_{1:t})\, dx_1\, dx_2 \ldots dx_{t-1} \quad (6)$$

Estimating such full posterior contains all the information we need about the map and the pose or the path [29].

As we will see a trade-off between accuracy and computational costs (both time and memory) is required. So approximation is ubiquitous in real SLAM applications, obviously due to the high dimensionality of continuous variables and the large number of correspondence parameters required.

1.2.2 *Issues and open problems*

Trying to solve the SLAM problem with sensors uncertainty, combined with computational constraints have the consequence of raising up some issues and challenges.

REPRESENTATION ISSUES The final objective is to give to the robot a useful map where it can do high level tasks, so what is it necessary to represent and how? The possibilities to build a map fall between a set of landmarks (as example visual features) just for cite some works: [5],[4],[30],[13]; or a dense set of data (as example laser scan data), again to cite some: [12], [10], [23].

There are many ways to represent Probability Density Function (PDF), two of them are more popular than others: in Kalman's Filter based SLAM Gaussian distribution are assumed for all PDFs (somehow limited but easy to implement) [5], in contrast in Particle Filter (PF) based SLAM sampling is used to estimate a generalized PDF (more general but may require a lot of sample) [10].

UNSTRUCTURED AND DYNAMIC ENVIRONMENTS In real world applications a robot has to operate autonomously for hours or even days, travelling for kilometres into 3D environment. So there is a need of light-weight map with features that are invariant to robot motion (as example SIFT [18] features are scale invariant).

Another problem rise up when we face that real world evolve with time, it is dynamic. Not only due to other moving objects, that should be identified and isolated, but also the light changes during the day. Summarizing, maps should be invariant also to light, temperature, whether, . . . the list can be very long.

DATA ASSOCIATIONS We encounter data association issues when we try to associate data from sensors observations to a portion of the map or a specific landmark. Data association must be robust or instead spurious measurement will be associated improperly giving for example a wrong pose estimation.

LOOP CLOSURE Noisy sensors and uncertainty in estimation let grow the *drift* in time. So without a drift correction procedure, if a robot walk in circle, it will not be able to know it.

The loop closure is the solution to the drift problem. It is the correction of the robot path and the estimated position of map components. When the robot sense a place that it have seen before, it should review the whole map (or a part) to remove some of the accumulated error.

A successful SLAM algorithm should be able to close loops efficiently since the map can be very large and it may have to respect deadlines in a real-time operation. In recent years some attempt to the loop closure issue has been made with notably success also in very large outdoor environment [13] [4].

An interesting approach in loop closure is the one from Galvez [9]. They look for timing consistency before accept the loop closure. When they have a candidate set of possible loop closing location they take into account only the subset that satisfies temporal consistence with previously viewed locations.

FAILURE DETECTION AND RECOVERY If the track of features is lost due to motion blur, too fast motion, occlusions or not mapped dynamic object in the environment (and other problems like the Kidnapped Robot Problem) the construction of a consistent map or the estimation of the ego-motion can become infeasible.

Other problems rise if the sampling rate of an IMU is not sufficient to detect a peak, that can be caused by a robot accidental drop or a hit. The estimated ego-motion will consequently diverge from the real one. For these and other reasons a failure detection system must be prepared to have a robust SLAM.

After the detection of the failure event the system should recover itself without loosing past information like the map. In some sense we can see it as the start of a new map that has to be merged to the old one. This approach can be found in works like the one from Eade and Drummand [6].

Another approach has been developed by Klein and Murray in [17] where in their key-frame-based SLAM they use directly the key-frames sensed after a failure detection. They compare them with all the key-frames stored in the map and relocate the robot into the most similar one.

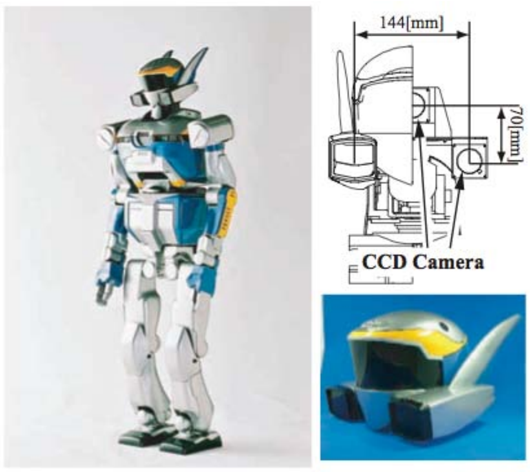
1.3 HUMANOID ROBOTS

The first key point that distinguish state-of-the-art works on SLAM for humanoid robots is the generation of the data. Data can come as examples from sensed frames from a video system or range data from laser scanners or inertial informations coming from an inertial sensor like an IMU. In following sections I will introduce them after a brief introduction on common humanoid robots.

1.3.1 *Common robots*

In the last few years, humanoid robots have become a popular research tool. They are assumed to offer new perspectives compared to wheeled vehicles since they are, for example, able to access different types of terrain and climb stairs. More generally, their human-like body can help when acting in a world designed for humans. The drawback of humanoids is that several tasks that can be easily carried out with wheeled robots are hard to achieve with legged systems. In such tasks are included stable motion with a payload and the accurate execution of motion commands. Even motion planning can be very hard due to the high level of degree of freedom that a humanoid robot has.

Since they have to simulate humans, a humanoid robot should have a stereo vision system on its head. So as consequence the majority of SLAM approaches for humanoids are vision based.



(a) HRP2 vision system

(b) SONY QRIO

(c) NAO with a laser rangefinder on his head

(d) Non-robot configuration. An IMU-camera system on the hardhat of an operator.

In fig. 1a you can see a quite diffused robot, the *HRP2*. It has comparable size as a human, 154*cm* height and 58*kg* weight. HRP-2 has a stereo vision system composed of three cameras(Fig. 1a upper right). Two horizontal cameras are separated by 144*mm*, and the third camera is 70*mm* upper than them. Relatively short focus lens is used whose standoff is from 0.5*m* to 4*m*. The shutter speed of the camera is controllable by a computer to adapt various lighting condition. Through a plastic shield in front of the cameras (Fig. 1a lower right), the object image is distorted. It is practically difficult to model the shield shape and the position of the camera precisely. So it's used to remove the shield from the robot [8].

Other robots have very similar configurations like QRIO (fig. 1b) from SONY. This is a small size humanoid robot. QRIO stood approximately 0.6*m* tall and weight 7.3*kg*. QRIO's slogan was "Makes life fun, makes you happy!". Sadly on January 26, 2006, on the same day as SONY announced its discontinuation of AIBO and other products, it announced that it would stop development of QRIO.

Recently some robots get updated with a small laser range finder collocated on their head, as example you can see the NAO robot in figure 1c. This open new prospective on SLAM for humanoid robots because data from such devices are very accurate. Oßwald and colleagues presented an approach with this robot configuration to climb a spiral staircase. They proposed an improved localization of the robot using both range and vision data [25]. The laser scanner resulted to be extremely accurate on the floor plane but quite noisy when climbing stairs due to motors drifts on robot motions and to hardware constraints that do not enable the scanner to see the area just in front of robot feet. So a combination of laser data, vision and a preloaded model of the stairs was used to achieve such good results in estimating robot location.

A quite diffused way is the *non-robot* configuration. Since humanoid robots are very expensive or since most of them are not really ready for real world operations, quite often researchers opt to simulate an humanoid robot with a real human. Another motivation is that some tasks like vision are not very related to the real cinematic of the robot so having a real humanoid robot sometimes is not strictly required. For example see figure 1d for a video system attached on a hat and other sensors in a backpack. This is taken from the work of Soatto, Pretto and Tsotsos [30], they used this non-robot configuration to estimate the ego-motion using a singular camera with an attached IMU.

2

PERCEPTION

2.1 PERCEPTION

The basic thing that a robot, or a general autonomous system need is the knowledge about its environment. To gather such knowledge, if not hard-coded, robots must rely on their set of sensors and acquire data from them. Raw data isn't really useful so an intermediate task of data elaboration and/or noise reduction is required (see section 3.1).

It's possible to classify sensors in base of their characteristics [26]:

PROPRIOCEPTIVE a sensor is *proprioceptive* if it measures values internal to the system (ex. the battery voltage);

EXTEROCEPTIVE a sensor is *exteroceptive* if it in contrast measures values from the external environment (ex. vision);

ACTIVE a sensor is *active* when it emit energy into the environment and than measure the response (ex. a sonar);

PASSIVE a sensor is *passive* when it measure the environment energy that enter the sensor (ex. a microphone).

As you saw in section (1.3) in state of the art humanoid robots have only three main type of exteroceptive sensors: *vision* (in stereo or monocular way), *laser scanners* and *IMUs*. In [7] its possible to find a large description of a very lot of other sensors.

2.1.1 *Laser scanners*

A Laser scanner or, laser range finder or lidar is a time-of-flight sensors. They consist of a transmitter that emit a collimated beam and a receiver capable of detecting the component of light coaxial with the transmitted beam. They estimate the distance of a point calculating the time that the beam need to come back to the receiver. They are 2D or 3D, for this scope a mechanical system is used to rotate in one or two axes a mirror to cover the whole scene. They are very expensive due to such mechanism and to the need of very fast electronic boards.

2.1.2 *Inertial Measurement Units*

Into an IMU there are a set of (three orthogonal) accelerometers and (three orthogonal) gyroscopes used to estimate its relative 6-DoF pose (*x*,*y*,*z* for position and *roll*, *pitch* and *yaw* for orientation), velocity and acceleration.

The gyroscope data is integrated to estimate the orientation, while accelerometers estimate the instant acceleration. The knowledge of the gravity vector make possible to transform such acceleration relative to it and after an integration the velocity is known. Integrating again will give the estimation of the position. The only thing that is missing is the value of the initial velocity, this can be assumed as zero. IMUs are high affected to drift, drift in gyros cause a wrong estimation/subtraction of the gravity vector that will rise a quadratic error in the position due to the double integration.

Since all IMUs will drift after a while they need to be coupled with other system that externally time by time correct their estimation.

In order to have good results IMUs often come with a calibration matrix that contains information about wrong axes displacement of accelerometers and gyros. This calibration is very expensive and each IMU needs its calibration, this is the difference between expensive and calibrated IMUs versus cheap and uncalibrated ones (the difference in price can vary in some order of magnitude).

2.1.3 *Vision*

As human being, we choice vision to be our favorite sense, so it tend to be the same when thinking about humanoids. With the enormous amount of data regarding the environment provided by vision we can infer high level informations necessary to do high level tasks. Actually video systems are ubiquitous in the world that surround us and videocameras can be very cheap. It's no surprising the fact that every humanoid robot have a video system.

IMAGE FORMATION A lot of work has been done in the fields of image capturing, image processing and image analysis; the last two combined are known as *computer vision*. Such works gave us the *pinhole camera model* and a *general perspective projection model* resulting in the following relation between coordinates of a pixel in an image and its 3D counterparts:

$$\lambda x' = K\Pi_0 g\mathbf{X}_o \tag{7}$$

Where $\lambda \in \Re_+$ is a scalar parameter, $x' = Kx$ are the calibrated image coordinates, K is the camera calibration matrix and x are the uncalibrated image coordinates, Π_0 is the standard projection matrix $[I, 0]$ from $\Re^3$ to $\Re^2$, g is the euclidean transformation from the camera frame to the world frame, and X_0 are point's coordinates [19].

CORRESPONDENCE PROBLEM The above relation let us become capable of image formation and skip to image analysis, where from a series of images consequent in time we search for features and try to track them during the image flow in time. As stated in [19]:

> The *correspondence problem* consists in establishing which point in one image corresponds to which point in another, in the sense of being the *image* of the same point in space.

Here image stand for being a pixel generated by a point in space through the pin-hole camera model.

Normally images in the flow are taken each from a different vantage point, so some computation has to take place in order to achieve feature matching. The intuition is that we have to model all possible transformation (translation, affine transformations, projections) that a feature can be affected during the image flow, in other words the motion model of that feature.

Saying that $I_i(\mathbf{x})$ is the image of a point x into picture i, we have to choose a class of transformation h and then we can approximate the matching of a feature from two pictures as:

$$I_1(\mathbf{x}) = I_2(h(\mathbf{x})) + n(h(\mathbf{x})) \tag{8}$$

where n is an additive noise term (occlusion were not taken into account).

So the formulation of the correspondence problem easily become the solution of an optimization problem [19]:

$$\hat{h} = arg \min_h \sum_{\tilde{\mathbf{x}} \in W(\mathbf{x})} \|I_1(\tilde{\mathbf{x}}) - I_2(h(\tilde{\mathbf{x}}))\|^2 \tag{9}$$

Here we have to look for the particular transformation $\hat{h}$ that minimize the effect of noise, subject to equation (8) integrated over a window W.

The choice of how to compute the discrepancy criteria, in this case the norm, is critical in the quality of the results and the computational effort needed for the calculation.

This approach is stated for point features, but it can be straightforward expanded for other geometric features like lines, edges or corners and also for high level features like doors, tables or patch features like Harris or SIFT. When it is necessary to work on features that cover a patch of pixels, there are various criteria applicable. As examples there are: the Sum of Absolute Difference (SAD) which just iteratively sum the subtraction pixel by pixel of the patch I_2 from I_1, the Sum of Squared Difference (SSD) which is very similar but uses more multiplication operations by squaring the differences before aggregation; and there is the Normalized Cross Correlation (NCC) that is more complex because it involves multiplications, division and square roots operations. As the computational complexity increase also the robustness of method increase in fact NCC is more distinctive from SAD and SSD in case of affine intensity changes [19].

FEATURE EXTRACTION PRINCIPLES With the objective in mind of creating a map that is useful to the robot, that is as general as possible in terms of environmental dynamics (light, temperature, . . .) and also invariant in the point of view, some principles must be taken into account in the feature extraction procedure.

3

DATA

3.1 FILTERING TECNIQUES

Filtering techniques are applied to have a better estimate of the state of the environment and of the robot due to the fact that raw sensors data is unreliable and unprecise taken alone.

Algorithm 1: Bayes filter algorithm

```
  Input: $bel(x_{t-1}, u_t, z_t)$
  Output: $bel(x_t)$
1 forall the $x_t$ do
2     $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$
3     $bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)$
4 end
5 return $bel(x_t)$
```

All filtering algorithms in probabilistic robotics are based on the *Bayes filter* that is the recursive algorithm (algorithm 1). It takes as input the belief state at time *t-1*, the control at time *t*: u_t and the measurement at time *t*: z_t. The output is the belief state at time *t*: $bel(x_t)$.

The *update rule* of Bayes filter algorithm consists on two steps: the *prediction* (line 2 of the algorithm) and the *measurement update* (line 3 of the algorithm). In the prediction step the filter try to estimate the belief only using information on the previous belief $bel(x_{t-1})$ and the control applied to the robot, so it consists of an integration (or summation) of the product of two distribution: the conditioned probability $p(x_t|u_t, x_{t-1})$ and the previous belief. The subsequent step refine the prediction using the information of actual measurement. For each possible x_t it multiples the previous prediction by the conditioned probability to see measurement z_t assuming to be in state x_t: $p(z_t|x_t)$. The result is finally normalized to guarantee that it is a probability, it must be that: $bel(x_t) \in [0, 1]$.

As is visible, the belief respect the Markov assumption (see sec. 4.1) in which it completely represent the past of the robot.

In a real application Bayes filter cannot be used because it cannot be implemented on a digital computer and approximation is mandatory also to maintain computational costs under control.

3.1.1 *Kalman filter*

The *Kalman filter* is a class of implementations of the Bayes filter that is applicable on continuous linear Gaussian systems.

Algorithm 2: Kalman filter algorithm

```
  Input: $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$
  Output: $\mu_t, \Sigma_t$
1 $\overline{\mu}_t = A_t \mu_{t-1} + B_t u_t$
2 $\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
3 $K_t = \overline{\Sigma}_t C_t^T (C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1}$
4 $\mu_t = \overline{\mu}_t + K_t(z_t - C_t \overline{\mu}_t)$
5 $\Sigma_t = (I - K_t C_t) \overline{\Sigma}_t$
6 return $\mu_t, \Sigma_t$
```

Gaussians can be represented with their moments (the mean μ and the covariance Σ). The implementation of the Bayes filter via such parameterzation is the Kalman filter (see algorithm 2). To have a correct estimate of the belief we need it to be gaussian. So three conditions must be met:

1. the initial belief must be gaussian;
2. the state transition probability must be a linear function with an added gaussian noise independent variable;
3. also the measurement probability must be a linear function with an added gaussian noise independent variable.

When a system meet these assumption it is called a *Gaussian system*.

In algorithm 2 it is explicit how the system is linear. Here x, u and z are vectors so also μ must be a vector and Σ must be a matrix. The rest of variables: A, B and C are matrices too. Q represents the covariance of the zero-mean noise that affect the measurement and R the covariance of the zero-mean noise that modelize the uncertainty of the state transition.

3.1.2 *Extended Kalman Filter (EKF)*

The Kalman filter can be extended to non-linear problems. This can be achieved by calculating the tangent to the non linear function that describe the probabilities. It is done by Taylor expansion of such function in a specific point obtaining the Jacobians of that non linear probability functions. Such Jacobians are used in the filter in place of the two non linear functions and the result is the EKF.

In lot of works like the one by Davison [5] in 2007 and previously by quite the same team in 2006 [24] EKFs were used to manage noise. This combination, EKF-SLAM, is defined by Thrun [29] as the earliest and the most influential SLAM algorithm.

3.1.3 *Particle filter*

Algorithm 3: Particle filter algorithm

```
   Input: $\mathcal{X}_{t-1}, u_t, z_t$
   Output: $\mathcal{X}_t$
1  $\overline{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2  for $m = 1$ to $M$ do
3      sample $x_t^{[m]} \sim p(x_t|u_t, x_{t-e}^{[m]})$
4      $w_t^{[m]} = p(z_t|x_t^{[m]})$
5      $\overline{\mathcal{X}}_t = \overline{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6  end
7  for $i = 1$ to $M$ do
8      draw $i$ with probability $\propto w_t^{[i]}$
9      add $x_t^{[i]}$ to $\mathcal{X}_t$
10 end
11 return $\mathcal{X}_t$
```

The Particle filter (algorithm 3) is an another kind of an approximated solution of the Bayes Filter. The key idea is to represent the posterior density functions of the state by means of a set of M particles $x^{[1]}, x^{[2]}, ..., x^{[M]}$. Each particle represents an hypothesis of the state (as examples the robot path): the denser a subregion of the state space is populated by particles, the more likely it is that the true state falls into that region [29]. The algorithm take as input the particle set at time $t-1$, the control and the measurement at time t. The first loop (lines 2-6) construct the respective $\overline{bel}(x_t)$ of the Bayes filter. The term $w_t^{[m]}$ is called the *importance factor*, it weights the sampled particle basing on the probability to see measurement z_t in that hypothetical state. Then the second loop (lines 7-10) resample the particles weighting their probability to be correct by the importance factor computed before. The output correspond to the $bel(x_t)$ of the Bayes filter.

You can see an example in fig. 1.

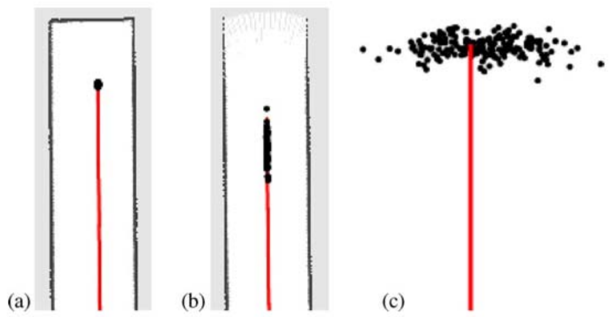


Figure 1: Particles distrubution in a corridor. a) future set; b) posterior set; c) raw odometry sampling in the same corridor.

3.1.4 *Rao-Blackwellized Particle Filter (RBPF)*

The use of Rao-Blackwellized particle filters by Grisetti in 2007 [10] and Kwak in 2009 [23] is one of the last approach to solve the SLAM problem with a 3D grid map. The structure of the SLAM problem enables particle filters to be applicable in real-time. The term 'Rao-Blackwellization' means factoring of a state into a sampled part and an analytical part. SLAM using RBPF computes the posterior over maps and a robot's path. The mathematical-key insight of RBPF-SLAM pertains to the fact that the full SLAM posterior can be factored as follows [29]:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1:t}, c_{1:t}) \prod_{n=1}^{N} p(m_n|x_{1:t}, z_{1:t}, c_{1:t}) \quad (10)$$

where $x_{1:t}$ is the robot path from the start up to time t, m is the map with N independent variables, and $z_{1:t}$, $u_{1:t}$ and $c_{1:t}$ are the measurements, controls and correspondences up to time t, respectively. In other words if correspondences are known each feature in the map is independent of each other.

So as stated before a particle filter is used to represent the posterior over some variables, and a Gaussian (like EKF) to represent all the other.

Montemerlo and Thrun introduced the Fast-SLAM method, [21]. Fast-SLAM is an instance of the RBPF and it uses particles to represent the posterior over the complete robot path and Gaussians to represent the posterior over landmark positions.

Grisetti et al. have developed this grid-based RBPF-SLAM generating a 2D map with a laser finder [10]. Kwak extended the Grisetti work with the stereo vision of an humanoid robot and a scheduling of grid matching to reduce computational cost [23].

4

MODELS

4.1 PROBABILISTIC MODELS

As an engineer the first thing to do after the statement and the understanding of the problem is the definition of the models of the systems involved in the solution to the problem. In our case: a motion model of the robot for ego-motion estimation, a set of sensors models and a data association model for mapping.

The choice of the models have repercussions in all consequent steps that take us to the solution.

The goal of these probabilistic models is to accurately describe the uncertainty of components like the motion actuators, the measurement read from sensors and the correspondence problem.

In practice the following models some times overestimate this uncertainty leading to algorithms that are very robust to infringements of the strict Markov Assumption.

Managing this assumption is fundamental in probabilistic robotics since it is the basis of the Bayesian Filter (presented in section 3.1) which is the basis of all the following probabilistic algorithms and models.

A system is on the Markov assumption if past and future data are independent if one knows the current state x_t [29].

So violations can be due to:

- unmodeled dynamics;
- inaccuracies into probabilistic models;
- approximation errors.

Most of these problematics can be resolved with an integration in the probabilistic model at the drawback of major computational costs and a major complexity in the implementation of the model.

4.1.1 *Probabilistic motion models*

The probabilistic motion model describe the conditional probability that the robot is in state x_t after control u_t from state x_{t-1}:

$$p(x_t|u_t, x_{t-1}). \tag{11}$$

The variable of a motion model is the pose of the robot relative to a world frame. The pose is composed of six elements: three Cartesian coordinates T (x,y,z) and three Euler coordinates for rotation Ω (yaw, pitch and roll).

We need to estimate the robot ego-motion and since the real motion is affected by noise therefore we should add noise variable to the system. If we can read the velocities of the robot from sensors (from an IMU or from an estimation from a camera or odometry) a simple model is the following:

$$\begin{cases} T_{t+dt} = T_t + (V_t + n_{V_t})dt \\ \Omega_{t+dt} = Log_{SO(3)}(R(\Omega_t)R((\omega_t + n_{\omega_t})dt)) \end{cases} \tag{12}$$

Where n_{V_t} and n_{ω_t} are Gaussian noise. This model results in practice good and robust lots of situations. The case that makes this model fails is when higher order derivative of velocities have correlation structures. This is the case of walking gaits in humanoids robots, or when the robot hits something. The following model includes these problematics and it is robust in real application to such walking gaits.

Jones and Soatto in [13] presented a model to estimate in real time the motion from monocular visual and inertial measurements. They used a random walk with four level of differentiation in the translation states as motion model:

$$\begin{cases} T_{t+dt} = T_t + V_t dt + \frac{1}{2}a_t dt^2 + \frac{1}{6}\xi_t dt^3 \\ \Omega_{t+dt} = Log_{SO(3)}(R(\Omega_t)R(\omega_t dt)R(\frac{1}{2}w_t dt^2)) \\ V_{t+dt} = V_t + a_t dt + \frac{1}{2}\xi_t dt^2 \\ \omega_{t+dt} = \omega_t + w_t dt \\ a_{t+dt} = a_t + \xi_t dt \\ w_{t+dt} = w_t + n_{w_t} \\ \xi_{t+dt} = \xi_t + n_{\xi_t} \end{cases} \tag{13}$$

Where t is the time index, dt is the time interval between discrete steps. V and ω are respectively the translational and the rotational velocities. a and w are translational and rotational accelerations. ξ is the translation jerk (the derivative of the translation acceleration). And finally n_{w_t} and n_{ξ_t} are two white noise process.

As described in [19], $R(\Omega_t) = exp(\widehat{\Omega_t})$ is the rotation matrix corresponding to the rotation vector Ω_t, where $\widehat{\Omega_t}$ is the skew-symmetric matrix corresponding to Ω_t, and $Log_{SO(3)}(R(\Omega_t)) = \Omega_t$ is the rotation vector Ω_t corresponding to the rotation matrix $R(\Omega_t)$.

In addition to this motion model, their work includes as variables the camera-to-IMU calibration and the gravity vector.

They introduced the concept of a *sufficiently exciting* motion sequence that is required in order to have a good estimation of the trajectory. The constraint is that the motion undergone by the sensing platform has non-zero linear acceleration and non-constant rotational axis. Under this assumption both the gravity vector and the camera-to-IMU calibration become estimable. Unfortunately many relevant motions like planar motions or constant speed do not respect these conditions and since both variables are not inferable. In such situations they decided to saturate the filter to prevent from drift.

In 2012 Tsotsos, Pretto and Soatto applied the work to the specific case of ego-motion estimation on humanoid robots [30].

For humanoids the quasi-periodicity of walking gaits and accelerations due to contact forces cause a *correlation structure* in such accelerations.

The robot pose is modeled by integrating the unknown angular and linear velocity, and the velocity is integrated by the unknown acceleration witch is further the integration of the unknown jerk. It is visible that this absence of knowledge is propagated until an high order of derivative that can be represented as white noise. At this level gaits are well modeled.

This is the reason to include such high order derivatives, the jerk ξ, in the model. It enables the capture of such correlations.

The strength point of this work is its independence from robot dynamics and sensors calibration.

4.1.2 *Probabilistic measurement models*

The second part of the system that need a probabilistic model is the set of sensors that measure the environment. Measurement models contain details of how sensors take informations about the physical world. Such models are sensor specific, so a different model is needed for each different type of sensor. In probabilistic robotics these models have to capture the sensor's noise. A formal definition is the conditional probability to see a certain measurement in a specific position of the map:

$$p(z_t|x_t, m) \tag{14}$$

Where z_t is the actual measurement, x_t is the actual pose of the robot and m is the map. Most sensors, when queried, generate a lot of measurement values instead of just one, so we can name this set of measurement with an overscript:

$$z_t = \{z_t^1, \ldots, z_t^K\} \tag{15}$$

Where z_t^k is a single measurement value. The probability distribution become:

$$p(z_t|x_t, m) = \prod_{k=1}^{K} p(z_t^k|x_t, m) \tag{16}$$

with the assumption of independencies of the noise between each measurement. As for the Markov assumption this is true only for the ideal case.

The map is represented of a set of objects within their characteristics $m = \{m_1, m_2, \dots, m_N\}$. As we will see in section 6.1, maps are usually modeled in two ways: location based maps or feature based maps.

In location based maps data is volumetric, they represent each location in the world and also the free space, so here each location has a list of elements m_n. Each element m_n in the list contains a feature and its displacement in the world relative to the world frame. A feature based map instead keeps informations only on the available features at specific locations.

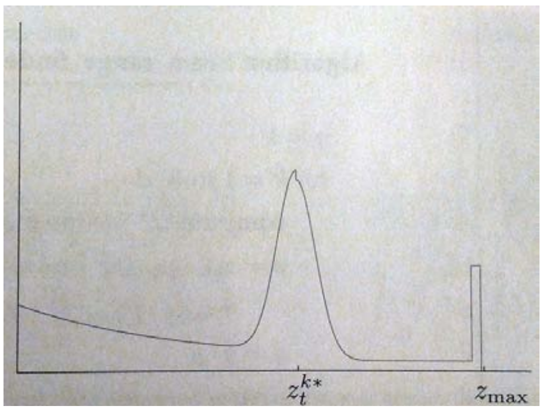


Figure 2: Example of the mixture model of a laser finder from [29]. The shape is formed by a gaussian centered on the true value z_t^{k*}, an exponential from 0 to z_t^{k*} for possible unexpected objects, a failure peak at z_{max} and a random uniform for random measures

The shape of the PDF (16) can vary a lot from different kind of sensor since each of them can have a different set measurement errors to model, so the resulting PDF will be a mixture model. This mixture model carry intrinsic parameters that must be set correctly in order to have a good estimation. These parameters can be set by hand, but a robust approach should use a parameter estimator. For example with a reference dataset $Z = \{z_i\}$ with associated position $X = \{x_i\}$ and a Maximum Likelihood (ML) estimator algorithm.

When a PDF is not known or its estimation can be exponential in time or more often when the independence assumption of equation (16) is not good; it is possible to approximate it to a known one or a simpler one.

The similarity metric between two distribution $P(Z)$ and $Q(Z)$ is the Kullback-Lieber Divergence (KLD):

$$D_{KL}(P, Q) = \sum_{x \in X} P(x) \log \frac{P(x)}{Q(x)} \tag{17}$$

where the sum is over all possible states in the distribution. This similarity is zero when the two distribution are equivalent or strictly larger otherwise [3].

The naive Bayes approximation is an approach to this problem with the strictly condition of independence between each variable on all the others.

In the case of vision, where locations are described by features, an high dimensional *discrete* probability distribution is generated. Chow and Liu trees is an approach to approximate discrete distribution that loosen this constraint of independence with the condition that each variable can be dependent on at most one other variable. This lead to a graph structure. In a graph with n variables there can be n^{n-2} trees and Chow and Liu in [2] developed a polynomial time algorithm to choose the best one. This technique is used in state of the art work FAB-MAP [3], [4].

Formally the Chow and Liu algorithm search the tree-structured Bayesian Network $Q(Z)_{opt}$ that minimize the KLD respect to the reference distribution $P(Z)$. The first step involve the construction of a mutual information graph $\mathcal{G}$. With n variables $\mathcal{G}$ is a complete graph where each edge (z_i, z_j) is weighted with the mutual information between z_i and z_j:

$$I(z_i, z_j) = \sum_{z_i \in \Omega, z_j \in \Omega} p(z_i, z_j) \log \frac{p(z_i, z_j)}{p(z_i)p(z_j)} \tag{18}$$

over all possible states of z. This value is zero for independent variables or larger otherwise.

Chow and Liu proved that the *maximum-weight spanning tree* in the mutual information graph is the resulting $Q(Z)_{opt}$. Doing this we exclude all minor dependencies between variables obtaining a fast tree structure where to efficiently search only for most important conditional dependencies between variables. In their work they proved also that the maximum likelihood can be obtained directly from co-occurrence frequency in training data.

4.1.3 *Probabilistic data association models*

Data association models are required when comparing measured location to the set of locations that compose the map. So these models should tell us if the actual location is a new one, so we should expand the map or if the robot is in a previous visited location and it has to perform the loop closure.

These models are mainly used in appearance based SLAM such as FAB-MAP [4]. In that work data is represented as *bag-of-word* in the sense that each scene is a collection of feature or *words* from a set or *vocabulary*. A scene at time k is represented as $Z_k = \{z_1, z_2, \ldots, z_{|v|}\}$ where $|v|$ is the size of the vocabulary and z_i is a boolean variable which is 1 if the feature i is present into the scene, 0 otherwise. They represent the map as a set of disjoint locations, with n_k locations at time k: $\mathcal{L}^k = \{L_1, L_2, \ldots, L_{n_k}\}$.

Each location L_j have an associated appearance model with extra variables e_i that describe the event that in that location exist an object that generates the feature z_i. So a location become the set: $\{p(e_1 = 1|L_j), \ldots, p(e_{|v|} = 1|L_j)\}$. A detector model is specified by:

$$\mathcal{D} = \begin{cases} p(z_i = 1|e_i = 0) & \text{false positve} \\ p(z_i = 0|e_i = 1) & \text{false negative} \end{cases} \tag{19}$$

The extra set of variables e let the system to easily merge data from multiple sensors with different error characteristics and it facilitates factoring the distribution $p(Z|L_j)$ into two parts. The first is a simple model for each location composed of independent variables e_i, and the second is a model that make use of Chow and Liu trees to incorporate correlations between detections of appearance words.

The objective is to estimate the robot position at time k given the set of measurement from time 1 to k, $\mathcal{Z}^k$. This is done by taking the location that maximize the conditioned probability:

$$p(L_i|\mathcal{Z}^k) = \frac{p(Z_k|L_i, \mathcal{Z}^{k-1})p(L_i|\mathcal{Z}^{k-1})}{p(Z_k|\mathcal{Z}^{k-1})} \tag{20}$$

it is a Bayes recursive estimation with three factors: the prior belief $p(L_i|\mathcal{Z}^{k-1})$, the observation likelihood $p(Z_k|L_i, \mathcal{Z}^{k-1})$ and the normalizing term $p(Z_k|\mathcal{Z}^{k-1})$.

The observation likelihood is computed using a Chow-Liu approximation, given the tree which is computed offline with training data, the conditioned probability become:

$$p(Z_k|L_i) \approx p(z_r|L_i) \prod_{q=2}^{|v|} p(z_q|z_{p_q}, L_i) \tag{21}$$

where z_{p_q} is the father of the node q in the tree. And including the e_i variables into the approximation:

$$p(z_q|z_{p_q}, L_i) = \sum_{s_{e_q} \in \{0,1\}} p(z_q|e_q = s_{e_q}, z_{p_q}, L_i)p(e_q = s_{e_q}|z_{p_q}, L_i) \tag{22}$$

A further approximation that involve independence between appearance words and locations and between $p(e_j)$ and words different from z_j makes the term $p(z_q|e_q = s_{e_q}, z_{p_q}, L_i)$ composed only of factors that are directly inferable from training data. The result is a very fast implementation of the approximation.

Another probabilistic approach that is used in feature based maps is the one introduced by Jones in [13]. In this work the map is divided in locations that are composed by nodes that share a portion of their set of feature together. They generate a query with the list of all the features that appear in a location and its neighbor and then use a scoring function based on the Inverse Document Frequency (IDF):

$$S_i = \sum_{k=1}^{K} \frac{f_q^k f_n^k}{\log N/F^K} \tag{23}$$

where N is the number of nodes, K is the number of terms, f_q^k is the number of times that the term k appears in the query list, f_n^k is the number of times term k appears in node i's list, and F^k is the number of documents in which term k appears at least once.

To have a metric on the location instead of the node, for each candidate location the score is the sum of all scores of nodes that fall in that location normalized by the total number of feature that appear in that location.

Without this division of locations in nodes the query will obtain results spread in the whole graph with many false positive, instead in this way only nodes with covisibility constraints should generate candidates.

After getting some candidates Random Sample Consensus (RANSAC) can select the best one. Jones et al. in their work make use of the gravity information from the IMU and with the fact that their co-visibility graph has an unique global scale it is possible to block some degree of freedom and to search exhaustively the right location in the candidate set.

5 LOCALIZATION

5.1 LOCALIZATION

The *naive* approach to the localization of the robot is to center the map in the robot position, delete features or voxels far away from it and maintain and updating only a window around the current position (Sabe et al. [14]).

Another approach is to build a complete map step by step in an incremental way and track the robot position in that map.

Sometimes the map is known a priori so the robot have only the task to localize itself in that map.

In a probabilistic map-based localization approach one should track probabilities of all possible positions of the robot. This is obviously due to the fact that sensors are affected by measurement errors and also by *aliasing*.

In order to achieve a good estimation of a body trajectory, or as usual in literature the robot ego-motion estimation problem, most common sensors used are inertials (like IMUs). Other approaches uses vision systems like mono-SLAM [5], visual odometry [28], visual SLAM [1] or FAB-MAP [4]. Furthermore mixture of sensors can be used like IMU-vision [13], laser range and vision [25] or more.

The tracking of the robot pose can be affected by drift problems when repeating the same operations like walking in circle.

5.1.1 *Probabilistic localization algorithms*

The straightforward implementation of the Bayes filter for the localization problem is the *Markov localization*. The only difference with algorithm (1) is that Markov localization needs as input also the map m and it uses it into the measurement model that become: $p(z_t|x_t, m)$; and into the motion model that become $p(x_t|u_t, x_{t-1}, m)$.

If the initial pose is known at $\hat{x_0}$ the starting belief is set to it:

$$bel(x_0) = \begin{cases} 1 & \text{if } x_0 = \hat{x_0}, \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

Instead if it is not known a priori it is set to a uniform probability distribution over all possible positions:

$$bel(x_0) = \frac{1}{|X|} \tag{25}$$

Where X is the set of all possible position x_0.

As consequence as being direct implementation of the Bayes filter, this algorithm is not useful in practice. In next paragraphs there are other algorithms that can be used for localization in real time applications.

As EKF is a special case of the Bayes filter, *EKF localization* is a special case of Markov localization. EKF localization represent the belief $bel(x_t)$ in a parametric manner with its first and second moments μ_t and Σ_t of a Gaussian PDF.

Particle filters are used in *Montecarlo localization*, where the belief is represented by a set of M particles $\mathcal{X}_t = \{x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]}\}$. The initial pose is distributed in M initial random particles from the prior distribution $p(x_0)$ and giving to them the same importance factor $\frac{1}{M}$.

The advantage of the use of the particle filter is that it can approximate every possible distribution, and we need only to increase the number of such particles to improve the approximation. As consequence the parameter M must being set as a trade off between accuracy and computational performances.

As stated in [29] a common approach is to continue sampling until an new pair of control and measurement arrive to the filter. In this way the system is adaptive to the amount of resource available. Obviously to avoid filter divergence a minimum number of particles is required.

Montecarlo localization can be modified to get some failure recovery capabilities. When the number of particles is small over a large volume of space is quite common that the filter will not be able to predict the correct pose. The idea is to generate random particles and add them to the particle set. This can help in case of the kidnapped robot problem, doing so let the set of particle converge in time to the new correct pose of the robot.

KLD-sampling (see section 4.1.2, equation (17)) is a variant of the Montecarlo localization. The objective of this variant is to determine the necessary number of particles measuring the approximation quality.

In the work by Oßwald et al. [25] the environment map is known a priori and they used range laser finder data with the Montecarlo Localization algorithm to estimate the robot pose (pose comprise the location and the orientation of the robot) over this map:

1. They transform the particle's pose according to the odometry information accumulated since the last integration. Starting from

this initial pose estimate, a scan matcher based on gradient descent improves the particle pose by fitting the current laser scan in the given 3D environment model.

2. As the most likely pose returned by the scan matcher already provides a good estimate of the robot's pose, they assume that the meaningful regions of the proposal distribution will be in the vicinity. Hence, they sample $\{x1^{(i)}, ..., x_k^{(i)}\}$ from a uniform distribution within a fixed range around the pose returned by the scan matcher.

3. At each sample point $x_j^{(i)}$, the algorithm evaluates the observation likelihood $p(l_t, C_t|m, x_j^{(i)})$ based on the current laser scan l_t and the set of camera images C_t. They assume that the laser measurement and the measurements from the individual images are conditionally independent.

4. A multivariate Gaussian distribution is fitted to the proposal distribution, from which it then draws the new particle poses in the resampling step because the algorithm can evaluate the proposal distribution only at sample points so it has to fit a continuous distribution to the sample values.

5. Then importance weights of the new particles are computed.

Results of such technique can be viewed in figure 3, on left the initial distribution and on right the final distribution with 95% of confidence inside the depicted ellipse.

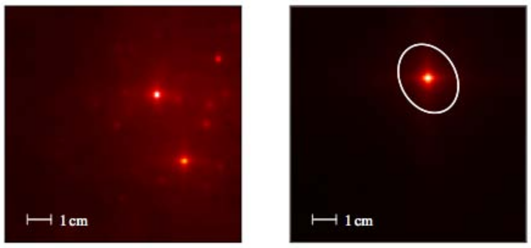


Figure 3: On left the initial distribution (observation likelihood of laser data) and on right the final distribution with 95% of confidence inside the depicted ellipse.

6

MAPPING

6.1 MAPPING

The most serious problem in describing the environment in 3D is that it consumes a large amount of memory and computing power. With the advance of computer technology, this restriction is relaxed in some degree. However it is still a serious problem for self-contained robots because of their size and weight limitation.

A number of systems assume the world is known in advance, i.e. they only work in simulation. Or sometimes severe restrictions took place to the environment or possible robot actions.

The simplest way to make a 3D map from range data can be using raw points as is or making polygons by connecting those points. But when simply accumulating maps, the amount of consumed memory increases. In addition, polygons equal to an obstacle, but it is difficult to distinguish a vacant space and a hidden space.

The analogous in visions system is to store each frame captured and merge all them to form a full detailed map.

The problem of construct a correct representation of the environment is somehow dual to the problem of the estimation of the robot position in that environment. In fact the definition of the specifics that describe the way that the map are being constructed have a great impact in how the robot pose is represented. Sometimes the accuracy of the robot position is limited by the coarseness of the map.

The grade of precision required is derived by the type of task that the robot has to do into the environment. And that grade, or the type of features that are used is also a consequent of the kind and the accuracy of the set of sensors equipped in the robot.

Maps can also be divided into two category basing on the manner to represent the environment. A map is *topological* if it uses a coarse graph-like to represents the environment. Nodes of such graph are rough sets features or in some recent works *locations* that include a superset of features that one might expect to see when visiting such locations. The other kind of maps are *metric* in that the environment is divided in regulary-spaced grids. This division of the space is not dependent on features displacement or shape.

Various approach has been developed in recent years to release such restrictions and I will discuss some of them in the following subsections.

The mapping problem in probabilistic robotics is to give the following estimation:

$$p(m|z_{1:t}, x_{1:t}) \tag{26}$$

Where m is the representation of the map, $z_{1:t}$ and $x_{1:t}$ are respectively the set of measurement and the robot positions from time 1 to time t.

6.1.1 *Plane classification*

Gutmann, Fukuchi and Fujita in their proposed method [12], they build a 3D environment map using occupancy grid and floor height maps. The resulting representation classifies areas into one of six different types while also providing object height information. The environment types that they wish to distinguish are designed for the navigation of the robot and are composed of types of regions where the robot can walk ordinarily (floor), require the robot to step up or down (stairs), allow the robot to crawl underneath an object (tunnel), are blocked (obstacle), are unsafe because the robot might fall down (border) or at last haven't yet been sensed (unknown).

They represent the environment by a regular 2D floor and obstacle grid map which is created from sensor data under the following assumptions:

1. The world is separated into floor and obstacles.
2. The floor is planar and horizontal (or else it is an obstacle).
3. There are no multiple floor levels at the same location.
4. The robot is able to distinguish between floor and obstacles and can estimate their relative position and height using its sensors.

Note that these assumption still allow the representation of obstacles above the floor where the robot can crawl underneath.

It is important that the floor height can be estimated precisely since kinematic constraints of humanoid robots need accurate information to ensure stable walking or climbing. On the other hand, obstacle heights are allowed to be imprecise since the only aim is to avoid them.

To classify voxels three parameters are needed:

OBSTACLE DETECTION: The highest occupied cell above a given floor height h_f defines the obstacle height at (x, y);

CEILING DETECTION: the lowest occupied cell above h_f defines the height of the ceiling at (x, y);

FLOOR CHANGE DETECTION: In the 8-neighborhood $U(x,y)$, the floor change $\Delta h(x,y)$ is the maximum height difference to the floor height $h_f\ Floor(x,y)$

So if a cell of the map has a ceil height higher than the robot (maybe in one of its pose) this cell is classified as a tunnel. If the ceil is low and the height is high the cell become an obstacle. If there is a small Δh the cell is classified as a stair and so on. For a real example see figure 4. A nice video of a real (not simulated) example can be found at `http://www.ijrr.org/ijrr_2008/096316.htm`

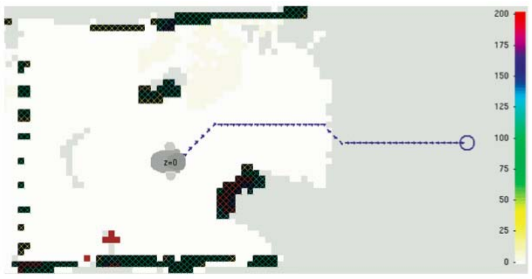


Figure 4: Map generated after the classify algorithm, QRIO walks around two obstacles.

6.1.2 *3D Grid map*

3D grid maps represent the environment in a fine grained grid over continuous space. The map m is composed of grid cells m_i: $m = \{m_i\}$, where:

$$m_i = \begin{cases} 1 & \text{if the cell is occupied} \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

And $p(m_i)$ is the probability that the cell m_i is occupied.

The mapping problem can be factorized in the following way:

$$p(m|z_{1:t}, x_{1:t}) = \prod_i p(m_i|z_{1:t}, x_{1:t}) \tag{28}$$

The price of this decomposition is the fact that dependencies between adjacent cells are ignored.

Obviously mapping algorithm should update only the set of grid cells that are effectively in the space measured by the set of sensors of the robot with an inversed sensor model that tell us the state of such cells based on the measurement.

Kanehriro et al. [8] in 2005 proposed a 3D grid map approach to the map generation. In their work a high resolution of the grid wasn't necessary because the map is built to find a movable space and not for object recognition (that need high resolution). The 3D grid map is generated after the SAD algorithm presented in section 2.1.3. A grid

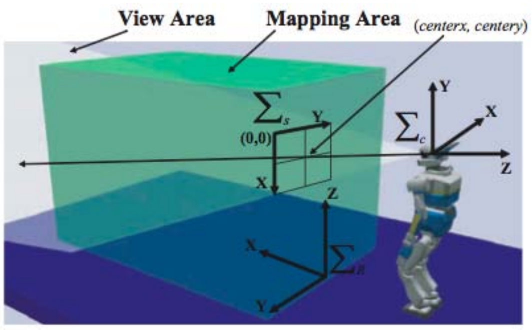


Figure 5: Coordinate definition.

point g_r is converted to the camera coordinate system $g_c(x, y, z)$ and then to the screen coordinate system $g_s(col, row)$ (figure 5).

If g_s is within the screen, the following procedure is executed.

1. If a 3D coordinate for g_s is calculated, Z coordinate of it and z are compared and g_r is labeled according to its result.
   a) If an absolute difference of them is smaller than a grid resolution, the grid is occupied by an obstacle. So it is labeled as Occupied.
   b) If z is larger, the grid is vacant. So it is labeled as Vacant.
   c) If z is smaller, the grid is hidden by an obstacle. So it is labeled as Hidden.
2. If a 3D coordinate is not calculated, the grid is labeled as Unknown.

You can see an example in figure 6. Map data is accumulated during the map generation.

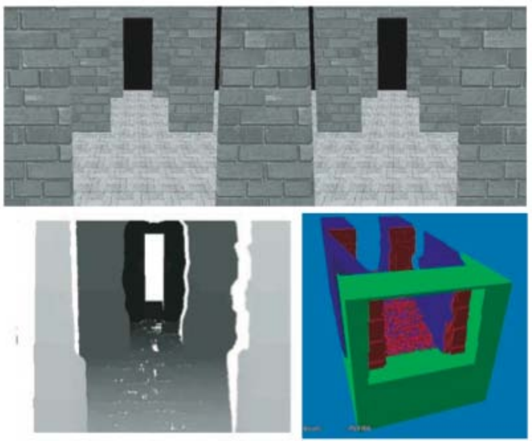


Figure 6: 3D grid map example.

Like in Kanehiro work, Kwak et al. used the same method but with a different approach on the voxel generation. They followed a statistical approach with a RBPF and a scheduling system for grid matching in data accumulation [23]. They made the following algorithm taking as input robot pose x_t and range data z_t and returning as output the updated particle set:

1. Predict the pose of the robot based on odometry.
2. If the range data is the first (nothing to match) register the data and return.
3. Else compute the importance weight and average likelihood. Compute Neff.
4. If computed values satisfy thresholds:
    a) Compute the means of poses and maps.
    b) Search the best pose using gradient descent method.
    c) Update the mean map with the range data at the best pose.
    d) Reset particles with the resulting data of matching.
5. Else update all the maps of the particles with the range data.

6.1.3 *Feature-based map*

The map is feature-based when features are extracted and tracked between images (fig. 7). Features are salient patches from raw data of a single measurement. Sometimes feature have 2D or 3D points associated to localize them in space.

Features are extracted from raw data that come from sensors, in humanoid robots the common feature based maps are constructed with visual features from the robot camera's frames.

One of the first work that uses a feature-based map in a SLAM approach is the Davison's one. In that work they assumes a Gaussian uncertainty for the whole state vector, therefore they maintain the mean and the covariance of the camera and features pose [5]. They based the estimation of the posterior density function over the state on a EKF.

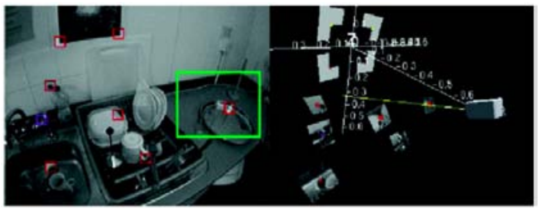


Figure 7: Feature-based map.

When they find a new feature they try to localize it with a 3D point: after the identification and first measurement of a new feature they draw a 3D line into the map along which the feature must lie. This is a semi-infinite line, starting at the estimated camera position and heading to infinity along the feature viewing direction. All possible 3D locations of the feature point lie somewhere along this line, the depth is not known due to the use of a single camera.

A set of discrete depth hypotheses is uniformly distributed along this line, which can be thought of as a one-dimensional probability density over depth, represented by a 1D particle distribution.

So they make the approximation of depth over the next few time-steps as this new feature is re-observed.

Once they have obtained a good depth estimate in the form of a peaked depth PDF, a conversion of the feature to 'fully initialized' with a standard 3D Gaussian representation is done.

A problem arise when the number of features is too high and on-the-fly decisions need to be made about when new features should be identified and initialized, as well as when it might be necessary to delete a feature. They found that with a wide angle camera 12 features are enough to localize the robot. This work resulted not good in scaling.

With the aim to develop high scalable maps, recent approaches made use of advanced storing techniques. BoW to limit the amount of necessary memory for each location, and the *inverted index* as a storing technique to obtain fast search performances in the place recognition phase.

The first step is to represent an image as the set of its most discriminative interest points. This is called *bag-of-features* technique [26]. Spatial information are lost but lot of space is saved and similarity between locations can be measured as the number of common features.

An improvement can be achieved with the *visual words* technique were each feature is viewed as a single number, namely a word. So the set of all words become a vocabulary. The vocabulary are constructed offline with train data and then used by the system [4], [9].

The construction is done by clustering similar features in a special feature discrete space (if the feature descriptor is as example a SURF 64bit, then the space will have 64 dimension). The space is then separated in Voronoi regions to divide clusters and one very distinctive descriptor for each cluster of features is chosen. That descriptor become the one that represent the cluster. The set of all cluster's descriptors become the vocabulary.

Doing this the dimensionality is drastically reduced and now fast comparison can be made. Obviously the number of words/clusters is crucial to achieve good comparison results. In literature (see [32] for a good comparison) there isn't any specific indication of the absolute best cardinality of a visual words vocabulary. A trade-off between discriminativity (more words) and generalizability (less words) must be choose.

New features extracted from test data are spanned into the discrete feature space and the closer cluster descriptor is taken in place on the original feature's descriptor.

With this schema, Z_k, the captured image at time k is stored in this type of BoW:

$$Z_k = \{z_1, z_2, \ldots, z_{|v|}\} \tag{29}$$

where $|v|$ is the number of words in the vocabulary and z_i is the number of word's of dictionary index i that are present in the image.

With binary-bag-of-words z_i is 0 if the word i is not present in the image, 1 otherwise [9].

Instead of storing a list of all locations with for each of them a list of the features present in that location, it is preferable to store in the inverse way, a list of all the features where each of them point to a list of all locations that contains that feature. This is called the *inverted file system* technique [33]. It is derived from the information retrieval field.

As summary I can state that the combination of the inverse index and the bag-of-visual-words is the state of the art golden standard for feature based maps.

6.1.4 *Loop closure*

All the sensors are affected by noise, so over time the drift from the ground truth can only increase. Nevertheless a robot should be able to recognize a previously visited location and hence correct its pose estimation. This problem is called the **loop closure detection**.

Davison et al. with their mono-SLAM [5] resolved such problem with a reasonable error (fig 8). As stated in section 6.1.3 this is one of the first approaches and it is not scalable as it work only for small loops and a few amount of features.

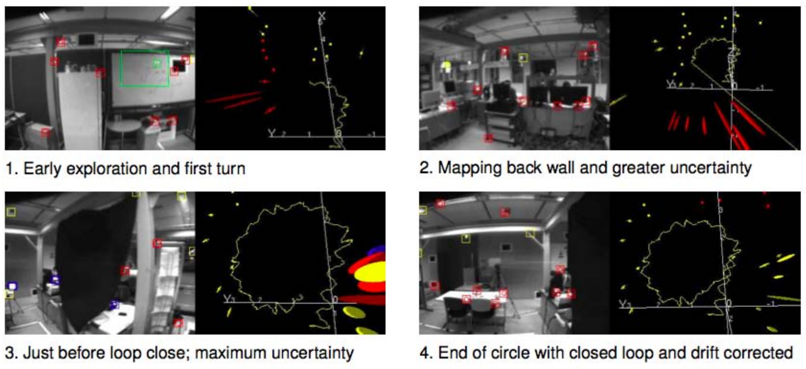

1. Early exploration and first turn
2. Mapping back wall and greater uncertainty
3. Just before loop close; maximum uncertainty
4. End of circle with closed loop and drift corrected

Figure 8: Example of HRP2 walk in circle.

They correct the position of the robot in the map when they find correspondence between new and old features.

In location based approaches the way to determine a loop closure is slightly different. Every time that a new location is sensed it is compared to the whole map looking for a match. This is done in feature-based map by extracting the new location feature set and comparing it with the older locations in the map.

The *inverted file system* technique (introduced in section 6.1.3) let the system take a new level of performances. It is used in most, if not all, of recent works as examples [13], [30], [4] and lot more.

In the case of visual words (introduced in section 6.1.3) the first index is the vocabulary of all possible words. The set of features of a sensed frame is quantized to the correspondent cluster descriptors set. The cluster descriptors are used to search all the locations that contain them. And a candidate set is constructed with the locations that contains much sensed features.

An interesting approach to skim the candidate set is the one from Galvez [9], they look for timing consistency before to accept the location. When they have a candidate set of possible loop closing location they take into account only the subset that satisfies temporal consistence with previously viewed locations.

When a candidate set is ready the common approach is to apply a geometric verification using RANSAC to select the best location [4].

In the work of Jones et al. [13] they didn't need to do that since they used an IMU-vision approach with a powerful model that cuts away some degree of freedoms like the gravity vector and the fact that they used a unique global scale. This let them to exhaustively examine each candidate with a number of test linear with the number of candidates, avoiding so the RANSAC approximation and its computational cost.

7 STATE-OF-THE-ART ALGORITHMS TESTS

7.1 LOOP CLOSURE ALGORITHMS TESTS

Since the development of self-driving cars gained a quite amount of attention in recent years (thanks also to the DARPA's challenges), the development of SLAM approaches got a boost.

A number of approaches to solve open problems came out like [29], [4] and lot more. Such approaches are made focusing on wheeled vehicles in outdoor environments. It would be appreciable if they can be applied straightforward to humanoids.

For this purpose I will test some of these loop closure approaches and a combination of them on an indoor dataset. Which is a typical environment for a humanoid robot that have to navigate inside a building.

The dataset consists of a loop of a corridor, starting and ending inside a small room. For screen-shots see figure 9.

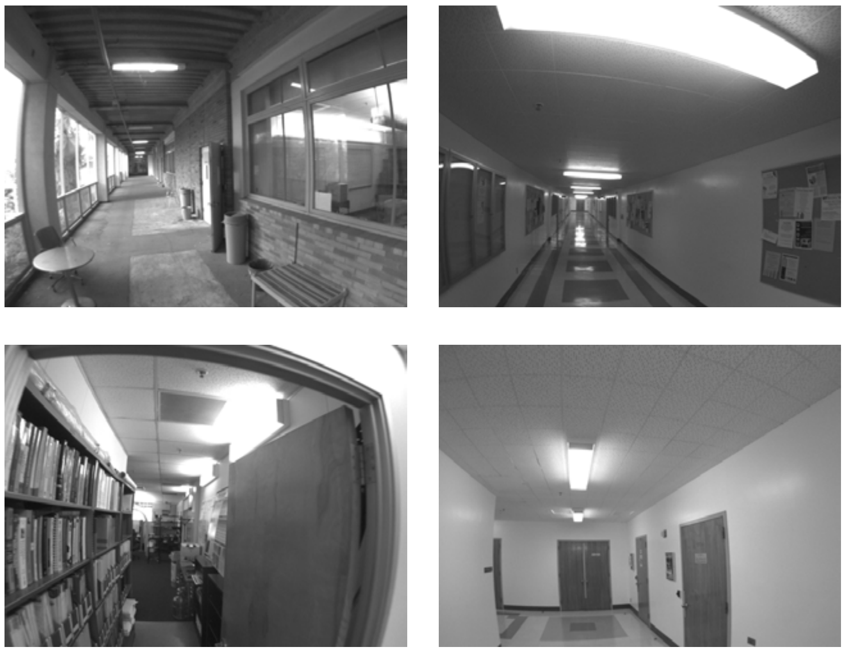


Figure 9: Dataset frames samples

The dataset is composed of 8436 frames of a video recordered at 33fps with the non-robot configuration in figure 1d in section 1.3, suited with a ground truth log of the camera pose inferred on the data collected by the attached IMU.

Checking the whole dataset against the map is way too expensive and relative useless because consecutive frames are really similar. So we used the pseudo-metric (equation 30) described in Whelan's paper [31] to skim frames:

$$m_{ab} = \|\mathbf{r}(P_{a_R}^{-1} P_{b_R})\|_2 + \|P_{a_t} - P_{b_t}\|_2 \tag{30}$$

Where $\mathbf{r} : \mathbb{SO}(3) \to \mathbb{R}^3$ is the function that provide the rotation vector form of some rotation matrix $\mathbf{R}$, and a and b are two poses.

The metric is sensitive to both rotation and translation movement. When a threshold is reached the relative frame is passed to the place recognition system. This distance is computed between the current frame pose and the last frame pose that has been passed to the place recognition system.

7.1.1 *OpenFAB-MAP*

FAB-MAP has become the golden standard for loop closure [9]. It has been released also as part of the OpenCV library (see `http://opencv.org`).

The work flow to obtain the loop closure prediction is well explained in the documentation and it is composed of the following steps:

TRAIN DATASET DESCRIPTOR EXTRACTION: the whole train dataset (composed of over a thousand of indoor images taken from the web) is iterated and for each image the features descriptors are extracted and stored into a matrix;

VOCABULARY CREATION: the set of descriptors are spanned in the descriptor space and clustered;

GENERATION OF TRAIN DATA: the whole train dataset is reiterated, the set of visual word are extracted from each image and stored in a matrix;

CHOU-LIU TREE GENERATION: using the above train data the tree is computed (see section 4.1.3 for details on Chow-Liu trees);

ON-LINE OR OFF-LINE RUN: the setup is now complete and the algorithm can now make the prediction, on-line if the frames are passed one at a time or as an off-line batch process if the images are passed all together.

The activity diagram in figure 10 depict the way you have to use the OpenFAB-MAP class of OpenCV. You have to iterate the whole test

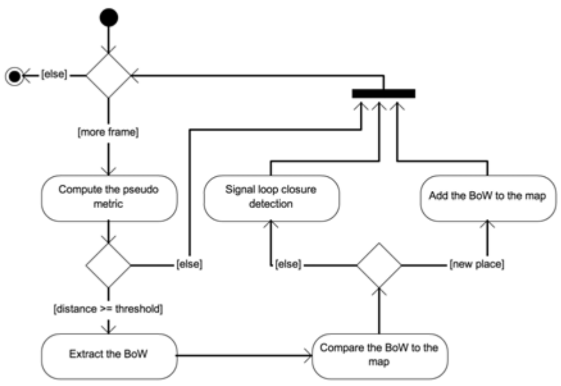
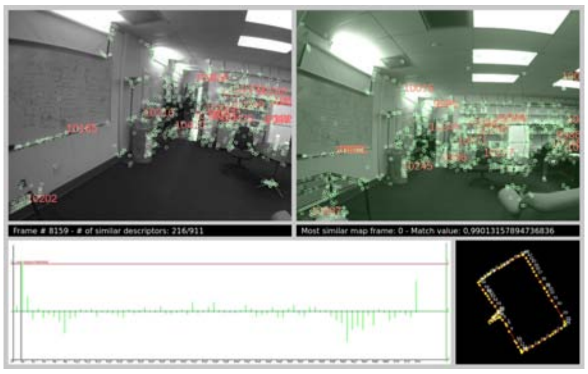


Figure 10: Open FAB-MAP 2 activity diagram

dataset, and chose which images pass to OpenFAB-MAP, for each of that images you have to compute the Bag-of-Words descriptor using the `cv::of2::BOWImgDescriptorExtractor` and check it against the map with the method `cv::of2::FabMap::compare(...)`. The results are stored in an vector of `cv::of2::IMatch` objects. The first element of the vector contains the probability that the frame is related to a new place. The rest are one object for each location in the map that contains the estimated probability that the frame is related to that location.

7.1.2 *GUI and results*

In order to have a better idea of what OpenFAB-MAP is doing frame by frame, we prepared a graphic user interface.



```
Frame # 8159 - # of similar descriptors: 216/911
Most similar map frame: 0 - Match value: 0.99013157894736836
```

Figure 11: Graphic User Interface sample

In figure 11 you can see how the GUI is structured:

FRAME VIEWS two frames are displayed, on left there is the actual frame, on right the most similar map's frame (green if the match is accepted, red otherwise);

INFORMATION DISPLAYS under the frames there are 2 small text displays that show useful informations like: the actual frame number, the number of similar descriptor of the two images, the index of the map's similar frame and the match value computed by FAB-MAP;

MATCH PLOT on the bottom left there is a plot of the matching vector, gray lines are the match values (peaks over the red line are accepted), green line are likelihood values. There is one line for each frame in the map plus one: the first is for the probability to have a new place;

GROUND TRUTH PLOT since the ego-motion estimation is provided we take it as the ground truth. It is plotted on bottom right, this is useful to see if matches are coherent. If not that means a false positive matches.

The red line is the camera path. Each yellow dot refer to a frame in the map. A dot is green if the actual frame is matched with a map's frame and that match is accepted as a loop closure. A dot is green if the actual frame is matched with a map's frame but that match is not accepted as a loop closure and the frame is added to the map as a new place.

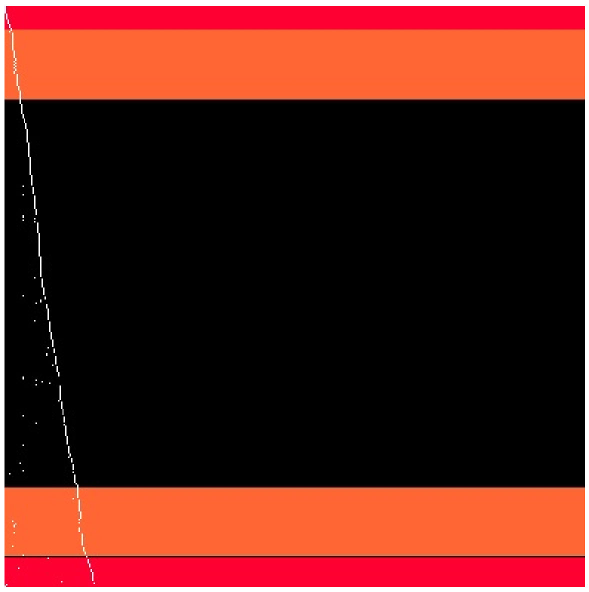


Figure 12: OpenFAB-MAP vanilla results

The figure 12 is a plot of the prediction result of a OpenFAB-MAP on-line run in the form of a confusion matrix. There is a row for each frame that is passed to OpenFAB-MAP. The white pixel is at the pseudo-diagonal if the frame is related to a new place in the map. If a loop closure is detected the white pixel is in the column of index equal to the matched frame map's index.

There are four coloured bands, two in red and two in orange. The red bands refer to frames related to the small room in which the loop starts and ends. The orange bands refer to frames related to the first corridor.

Loop closure detection are expected only in these two areas (not the whole bands but only the in the columns indices that correspond to frames of the same area).

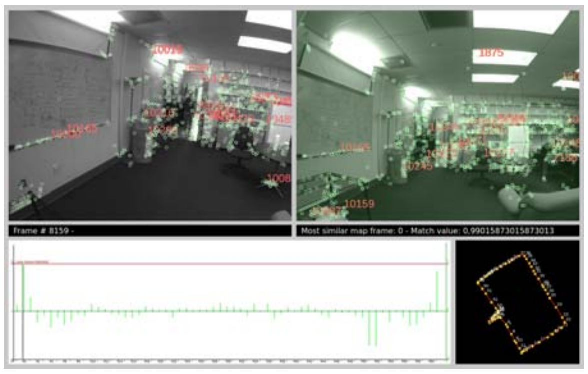
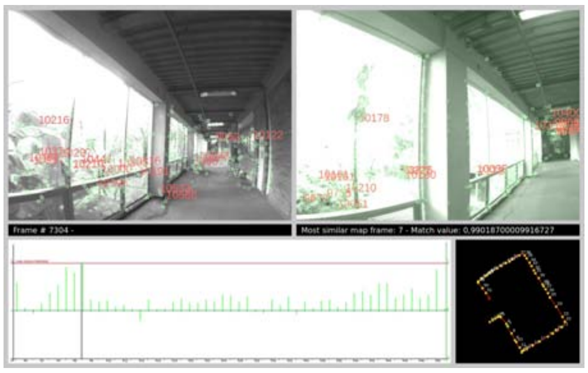



Figure 13: Two good match: a) in the room and b) in the corridor.

In figure 13 there are two good matches, one in the small room at the end of the dataset and one in the corridor outside the room when the camera is returning to the room.

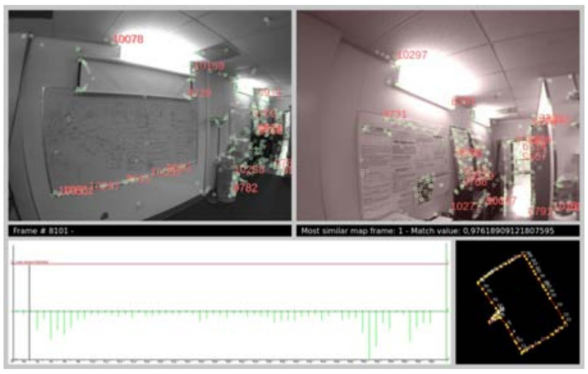


Figure 14: False negative match from the black band's frames

In figure 14 there is an example of a false negative loop closure detection, in this case OpenFAB-MAP failed to detect that the camera has returned inside the room.

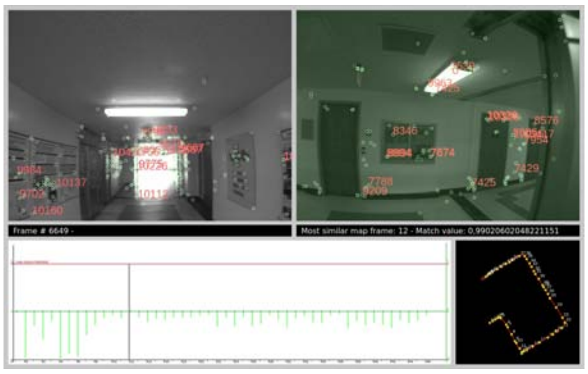

Frame # 6649 -
Most similar map frame: 12 - Match value: 0.99020602048221151

Figure 15: False positive match from the black band's frames

As you can see there are a lot of false positives, this is because FAB-MAP works only in the space of the appearance and it has not information on the path of the camera, in figure 15 you can see one of this false positive taken in the black band (the part of the loop that should not generate loop closures).

7.1.3 *Naive Vector of BoWs approach*

To have a comparison with another approach I decided to use a Naive Vector (NV) approach. The NV approach has two phases:

BoWs comparison: compare BoW descriptors in the descriptor space ($\mathbb{R}_{+}^{|V|}$ where $|V|$ is the size of the vocabulary) with the $L2$ norm. We then populate a candidate set of the k most similar BoWs;

Descriptors comparison: select the best candidate using a constraint in the similarity of the feature's descriptors between the actual frame and the frames related to the candidate set.

The feature's descriptors are compared using the Nearest Neighbour Distance Ratio (NNDR) described in [22]. With this metric the threshold is applied to the distance ratio between the first and the second nearest neighbour. Thus, the regions **A** is matched to region **B** if:

$$\|D_A - D_B\| < \epsilon \|D_A - D_C\| \tag{31}$$

where D_X is the descriptor of a region X, D_B and D_C are the first and the second nearest neighbor to D_A.

The parameter $\epsilon \in (0, 1]$ has to be decided empirically. Next to 1 if the constraint need to be permissive or next to 0 otherwise. As you see in next paragraph I tried three values for this parameter: 0.5, 0.65, and 0.8. Tests outside this range resulted useless.

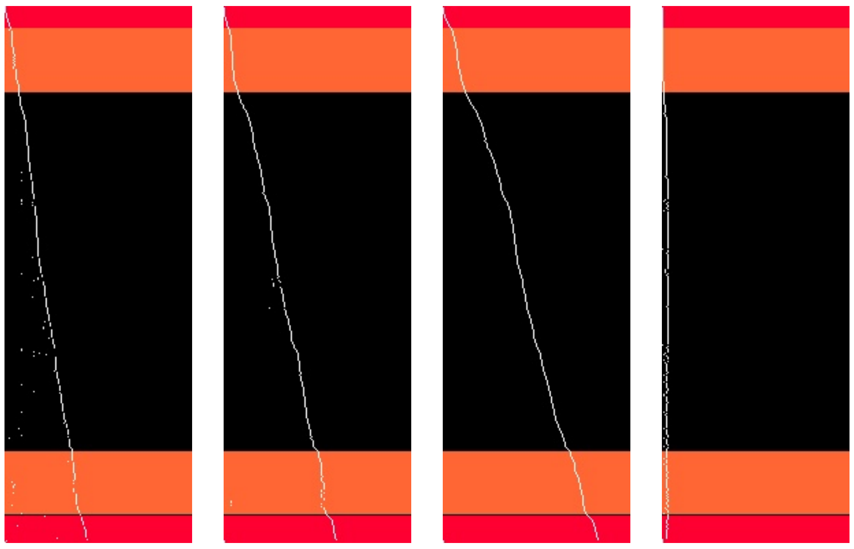


Figure 16: A comparison of approaches and parameters changes: a) OpenFAB-MAP vanilla; b) NV with NNDR 0.65; c) NV with NNDR 0.5; d) NV with NNDR 0.8.

In figure 16 there is the comparison between the confusion matrices of the pure OpenFAB-MAP and the NV approach with three different values for ϵ parameter.

The first thing that is visible is that for $\epsilon = 0.5$ the constraint is too restrictive and no loop closure is accept with the exception of the last frame which is very similar to the first one.

With $\epsilon = 0.8$ we obtain the inverse result, in fact the constraint now is too permissive and quite all the comparison result in a loop closure detection.

By setting the parameter $\epsilon = 0.65$ we obtained quite a good result. As you can see there is very few false positive matches comparing to the OpenFAB-MAP results, and the system is still capable to recognize both the corridor and the room.

7.1.4 *OpenFAB-MAP and* NNDR

Due to the hardness of the dataset in which al lot of location are similar each other, OpenFAB-MAP generates a lot of false positive loop closure.

To remove some of such false positive we decide to try to apply the NNDR constraint at the candidate that OpenFAB-MAP suggests as a loop closure.

In figure 17 there is a comparison between the vanilla version of OpenFAB-MAP (a), two with the NNDR constraint (b) and (c) and the best result obtained with the NV approach (d).

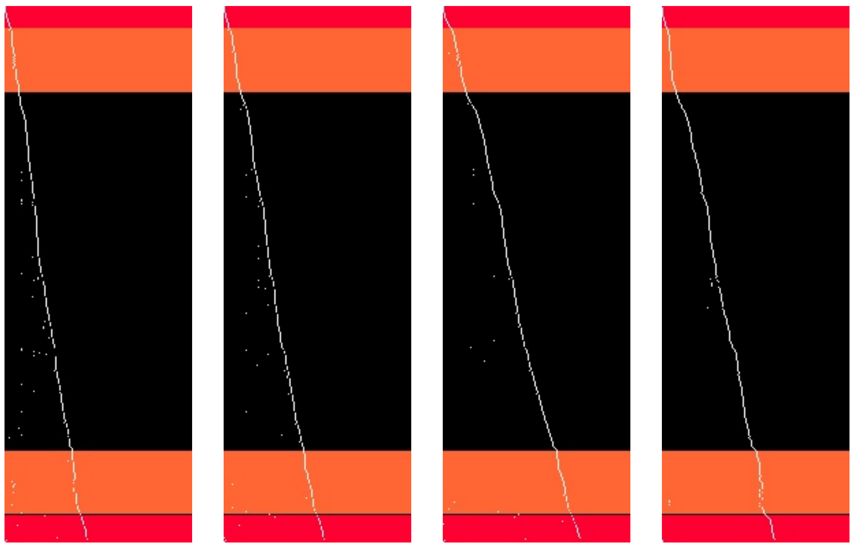


Figure 17: A comparison of approaches and parameters changes: a) OpenFAB-MAP vanilla; b) OpenFAB-MAP with NNDR 0.65; c) OpenFAB-MAP with NNDR 0.5; d) NV with NNDR 0.65.

As you can see lot of false positive are rejected, especially by setting $\epsilon = 0.5$.

The next step is to make use of the camera path estimation that is given with the dataset. It has been extracted from an IMU sensor attached to the camera. With spatial informations should be possible to reject other false positive loop closures detections.

Beyond the number of false positives, the system cannot recognize the loop closure in the entrance of the room. Especially the NV approach recognize the room well only when the robot re-seat on the chair from where the video started. It would be very nice if it is possible to recognize that place from the entrance. There, matches are very bad because feature descriptors are not robust to this severe viewpoint changes of the library and the posters.

8

A LOOP CLOSURE APPROACH ROBUST TO SEVERE VIEW-POINT CHANGES

8.1 MOTIVATIONS

Some problems raised up form the tests on FAB-MAP and NV approaches to the loop closure problem. For this reason we decided to exploit the information of the ego-motion estimation that is suited in the input data set.

In particular we focused on the difficulties that take that approaches to reject the loop closure detection in the first frames of the re-entering in the room, see example frames in fig. 18

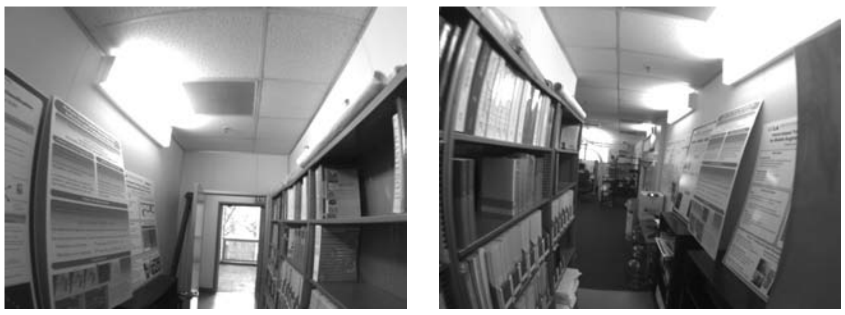


Figure 18: Sample frames of a desired loop closure detection. Frame left exiting the room, frame right re-entering the room.

The same problem appears also in [9] where loop closures are detected only when passing through a place in the same view-point verse. This can be solved or simplified using an omnidirectional camera but it isn't the case of our input dataset. The aim is to obtain features descriptors that are less invariant to such severe view-point changes. The best will be to have a frontal view of each feature [11].

By exploiting the ego-motion estimation informations one can locate the feature points in space by triangulation. The triangulation process need 2 video frames that see the same set of features and that are near in space to avoid large drift error of the ego-motion estimation.

Once a feature is located in space one can imagine to stretch and rotate the image frame to obtain an estimation of a frontal view of that feature.

The proposed approach is based on the strong assumption of planar features (see it as a derivation of the Manhattan assumption [16]). Under this hypothesis one can:

- estimate the plane where the feature point lays (or equivalently the plane's normal);
- take a neighbourhood around the feature point in 3D accordingly to the plane;
- re-project that set of points on the image;
- extract the related patch of pixels from the image.

That patch will be our estimated frontal view of the feature.

8.2 CHOICE OF KEY-FRAMES

The triangulation require the selection of couples of video frames. These frames should be chose related to poses that are distant enough to have a good parallax but not too far because they have to share a large portion of the scene in order to get a lot of features.

For this reason also the orientation is a crucial parameter. If the two frames are related to poses distant enough but one is orientated on left and the other on right, probably no portion on the scene will be shared by that frames and no features can be extracted and located in space by a triangulation.

Starting from the first, a frame is taken and labelled as *reference frame*.

The distance ($L2$ norm between relative poses) from the reference frame and current frame is measured and when a threshold (an input parameter) is reached the current frame is chose as support frame for the triangulation. The threshold must be set in order to guarantee enough parallax of the features viewed by the two different points of view.

Also the orientation is checked in order to ensure that the 2 frames will share a big portion of the scene. The orientation is checked versus a threshold (an input parameter) with the metric (considering only the rotation part) described in equation 30 in section 7.1.

As depicted in fig. 19, (P_A, P_B) are the selected couple of frames. From each frame features are extracted and then a match is performed to select a subset of features that are visible in both reference and current frames. Since these two frames are near in time and space, lots of features will be in this subset and will be good for triangulation. The fig. 20 is an example of a real couple of frames with the matched features highlighted.

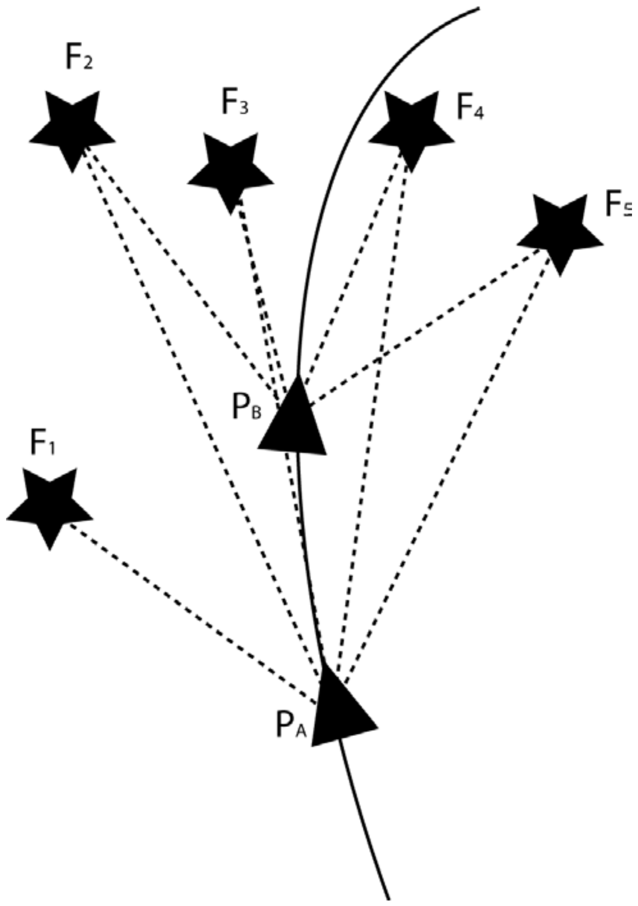


Figure 19: Triangulation of feature points by two views.

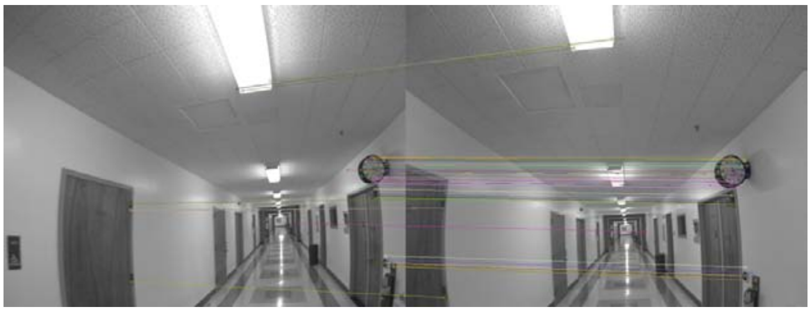


Figure 20: Matches of features extracted from a couple of time and space near frames.

If the cardinality of the subset of matched features is too low the couple is rejected and the current frame is taken as reference frame to build a new couple.

8.3 TRANSFORMATIONS INVOLVED IN THE TRIANGULATION

The input data of the *robot* pose is related to the IMU pose. For computation simplicity and consistency when we triangulate the feature points we want to put everything in the coordinate system of the reference frame. Due to how the camera-IMU system is built some transformations need to take place in order to obtain data consistent for our triangulation framework.

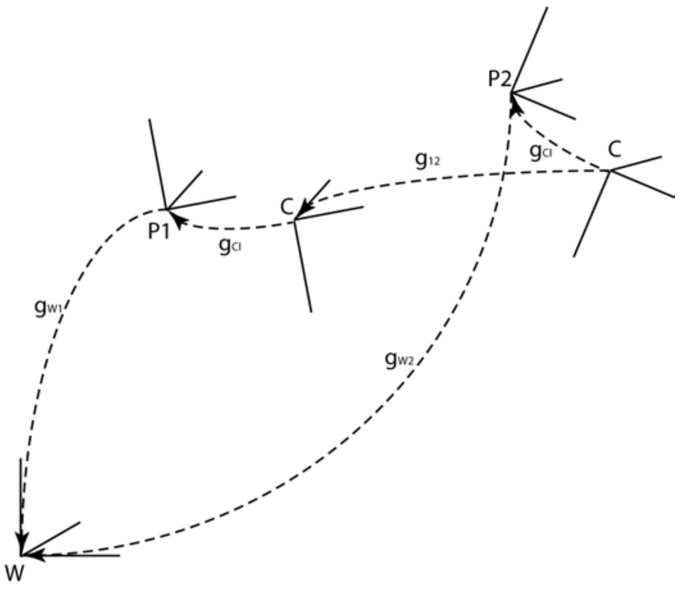


Figure 21: The transformations involved: **W** is the reference world frame, $\mathbf{P_1}$, $\mathbf{P_2}$ are the poses of the IMU in the two frames involved in the triangulation. $\mathbf{g_{12}}$ is the transformation from camera 1 to camera 2. $\mathbf{g_{IC}}$ is the transformation from the camera frame to the IMU frame. $\mathbf{g_{W_x}}$ is the transformation from $\mathbf{P_x}$ to the world frame **W**.

In figure fig. 21 such transformations are depicted in detail:

W is the world coordinate system;

$\mathbf{P_1}$, $\mathbf{P_2}$ are the reference and current IMU poses relative to **W**;

C are the poses of the camera;

$\mathbf{g_{IC}}$ is the fixed transform from the camera to the IMU frame;

$\mathbf{g_{W1}}$, $\mathbf{g_{W2}}$ are the transformations from the reference and current IMU poses to **W**;

$\mathbf{g_{12}}$ is the transformation from camera 2 to camera 1.

All poses in the ego-motion estimation log are in reference of the world frame W.

We take (an arbitrary choice) the camera pose of the reference frame as the coordinate system where we make al the computations. This choice affect the projection matrices that have to be passes to OpenCV for the triangulation.

The function `cv::triangulatePoints()` is made for stereo vision systems. If we know the transformation $\mathbf{g_{12}}$ from the camera 2 to camera 1 we can simulate a stereo vision system using our two views of the same scene. We can get it following the transformations chain in fig. 21:

$$\mathbf{g_{12}} = \mathbf{g_{IC}}^{-1} \cdot \mathbf{g_2}^{-1} \cdot \mathbf{g_1} \cdot \mathbf{g_{IC}}; \tag{32}$$

All the variables are known: $\mathbf{g_x}$ from the ego-motion estimation and $\mathbf{g_{IC}}$ is an input parameter.

With that transformation one can obtain a set of triangulated points. In fact the projection matrices become:

$$\begin{cases} \mathbf{P_r} = \Pi_0 \\ \mathbf{P_c} = \Pi_0 \cdot \mathbf{g_{12}} \end{cases} \tag{33}$$

Where Π_0 is the standard projection matrix $[I, 0]$ (as in sec. 2.1.3), P_r is the projection matrix relative to the reference frame, and P_c is relative to the current frame. These matrices can be passed to the OpenCV function for the triangulation.

The identity matrix is used instead of camera parameters (intrinsic and extrinsic) because the set of input points has been undistorted and modified as if they has been taken by an ideal camera.

This can be done in OpenCV by using the function `cv::undistortPoints (...)` that take as input: the camera matrix and the distortion coefficients; and return as output the ideal points.

8.4 PLANE ESTIMATION

The assumption of planar features is exploited in the successive step: the estimation of the plane of each feature. It is done by a minimization process. Taking a neighbour of pixel around the feature point in the *reference image*, projecting them on the 3D plane of the feature and then back project the set of 3D points in the *current image*. The residuals to minimize are the differences between the intensity of pixels in the current and the reference images.

The minimization is done with the support of the Point Cloud Library (PCL) for data management and visualization and the Levenberg-Marquardt least-squares minimization (lmfit) library for the minimization process. The optimization is done *one normal at a time*.

The minimization operates on spherical coordinates (ρ, ϕ, θ) in fig. 22 of the plane's normal. There are only 2 variable since $\rho = 1$ because normals are versors.

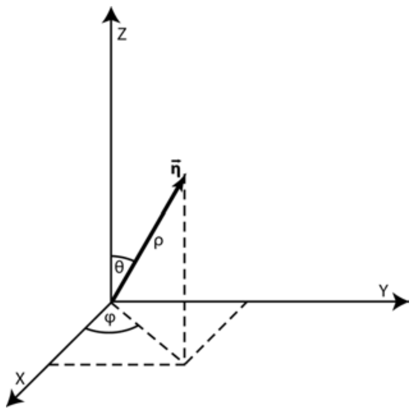


Figure 22: Spherical coordinates.

The process is done in a pyramid of k_p levels (k_p is an input parameter) where level 0 is the original image. At level i the image is a sub-sampled version of the original image by a 2^i scale (see fig. 23).

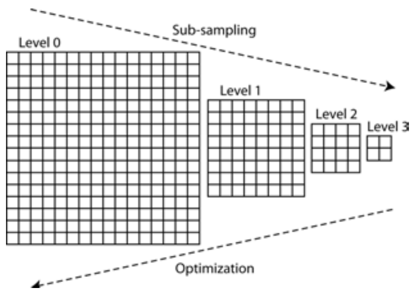


Figure 23: Pyramid's levels.

The process starts at the higher level of the pyramid where the image is very coarse, letting to make a first fast refinement of the initial guess. At the end of each level the result is passed to the subsequent as its initial guess. When the optimization finish to work on the level 0 the result is considered as our estimated plane's normal.

The normal's initial guess $\mathbf{n_{ig}}$ is taken for simplicity as the direction *feature-point to camera* as visible in fig. 24:

$$\mathbf{n_{ig}} = \frac{f_i}{\|f_i\|} \tag{34}$$

Where f_i is a 3D feature point considered as a vector attached to the origin. The resulting spherical coordinates are computed using the MATLAB like function `car2sph()` exposed in alg. 4. There is also the reverse version `sph2car()` exposed in alg. 5.

To compute the residuals a neighbour of pixels around each 2D feature point from the *reference frame* is taken, the diameter d_p is an input parameter (performances are very sensible to this parameter). These sets of pixels are projected in 3D using OpenCV functions on the respective planes and then back projected in the *current frame*. In

Algorithm 4: car2sph

```
Input: A versor $\mathbf{n}$
Output: $\theta, \phi$
$\theta$ = atan2( $\mathbf{n}_z$, sqrt($\mathbf{n}_x^2 + \mathbf{n}_y^2$));
$\phi$ = atan2( $\mathbf{n}_y$, $\mathbf{n}_x$ );
return $\theta, \phi$
```

Algorithm 5: sph2car

```
Input: $\theta, \phi$
Output: A versor $\mathbf{n}$
$\mathbf{n}_x$ = cos( $\theta$ ) $\cdot$ cos( $\phi$ );
$\mathbf{n}_y$ = cos( $\theta$ ) $\cdot$ sin( $\phi$ );
$\mathbf{n}_z$ = sin( $\theta$ );
return $\mathbf{n}$
```

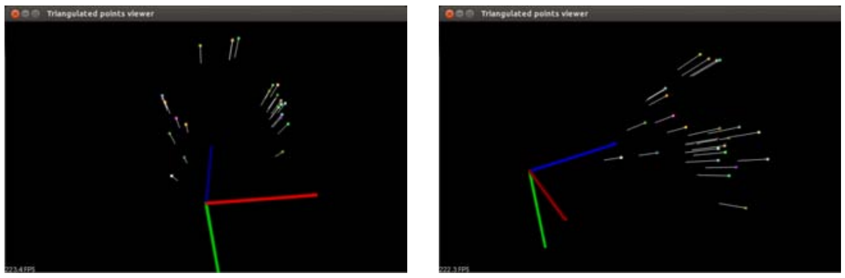


Figure 24: Initial guesses for normals.

fig. 25 it is visible an exaggerated example with diameter $d_p = 128px$ in reference frame on left and in the current frame on right.

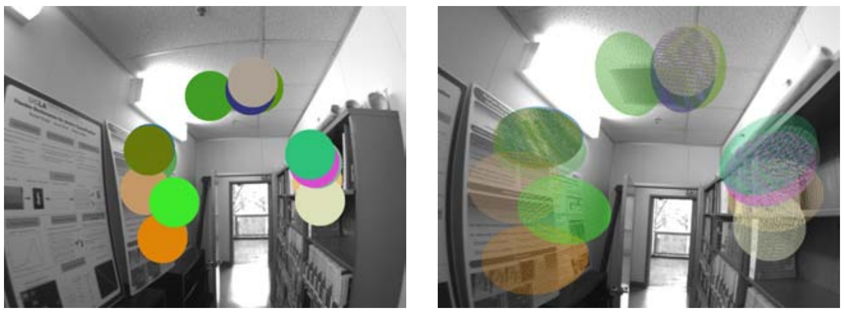


Figure 25: Projection of the reference frame (left) pixels in the current frame (right).

The lmfit library work with the Levemberg-Marquardt algorithm for the minimization a generic sum of squares.

In our case these squares are the difference between a pixel intensity in the reference $I_{i,r}$ frame and its counterpart in the current frame$I_{i,c}$:

$$sum\_to\_minimize = \sum_{i=0}^{N} \|I_{i,r} - I_{i,c}\|^2 \tag{35}$$

The parameters for the minimization are the spherical coordinates of the norm under process. To limit the variability of the parameters, in order to keep the normal near the initial guess of each level of the pyramid, a weight is added to each term w_i of the sum of squares to minimize:

$$\begin{aligned} \theta_{i,diff} &= \|\theta_i - \theta_{ig}\| \\ \phi_{i,diff} &= \|\phi_i - \phi_{ig}\| \\ w_i &= \begin{cases} 1.0 & \text{if } (\theta_{i,diff} < max_{diff} \text{ and } \phi_{i,diff} < max_{diff}) \\ e^{(1+\phi_{i,diff})\cdot(1+\theta_{i,diff})} & otherwise \end{cases} \end{aligned} \tag{36}$$

Where max_{diff} is an input parameter (as example $\pi/4$), θ_{ig} and ϕ_{ig} are the spherical coordinates of the initial guess of each level of the pyramid.

The sum to minimize with the weight will become:

$$sum\_to\_minimize = \sum_{i=0}^{N} (w_i \cdot \|I_{i,r} - I_{i,c}\|)^2 \tag{37}$$

In 3D, the points will results like in fig. 26. The estimated normals are visible in white.

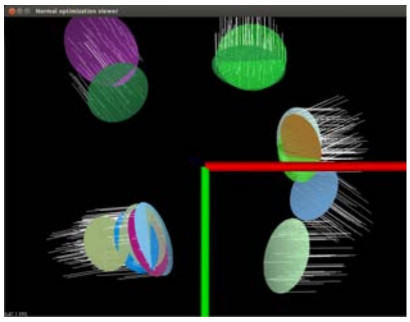


Figure 26: 3D projection of the reference frame pixels feature point's planes.

8.5 FEATURE EXTRACTION

In order to extract a feature descriptor one have to define the key-point of the feature. In OpenCV the key-point variables are:

- the point coordinates (the pixel in the image);
- the diameter of the meaningful key-point neighbourhood;
- the angle;
- the response (a value useless in our case);
- the octave.

With the estimated planes it is possible to extract a set of patches of images according to that planes. The coordinates are set right in the center of each patch.

It is possible to fix the *scale* degree of freedom for the descriptor extraction process. In fact with the patch it is possible to set the diameter as big as the patch width (the patch width is an input parameter, for example $32px$). And the octave is set to 0.

The least degree of freedom is the rotation angle that can be set to a fixed value (-1 for the OpenCV case) by computing it accordingly to the gravity vector. The gravity vector is known from the IMU data.

So we have: the plane normal and the gravity. And we have to append to the 3D feature point a coordinate frame.

The **Z** axis is set equal to the estimated normal. Considering the plane determined by the feature point, the normal and the gravity vectors it is possible to determine the **X** axis by setting it perpendicular to that plane as in fig. 27.

And consequently the **Y** axis must be perpendicular to both **X** and **Z**. Everything is recap in equation 38

$$\begin{cases} \mathbf{Z} = \mathbf{n} \\ \mathbf{X} = \mathbf{g} \times \mathbf{Z} \\ \mathbf{Y} = \mathbf{Z} \times \mathbf{X} \end{cases} \tag{38}$$

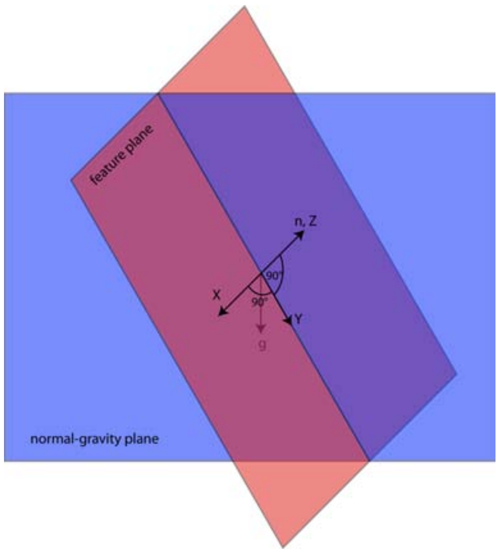


Figure 27: The coordinate frame attached to the feature point.

Now with no more degrees of freedom it is possible to extract a patch of a feature that will simulate a frontal view of the feature, which is also scale and rotation normalized.

In fig. 28 it is visible an example of a back-projection of the extracted patches in the reference frame. And in fig. 29 there are examples of that patches.

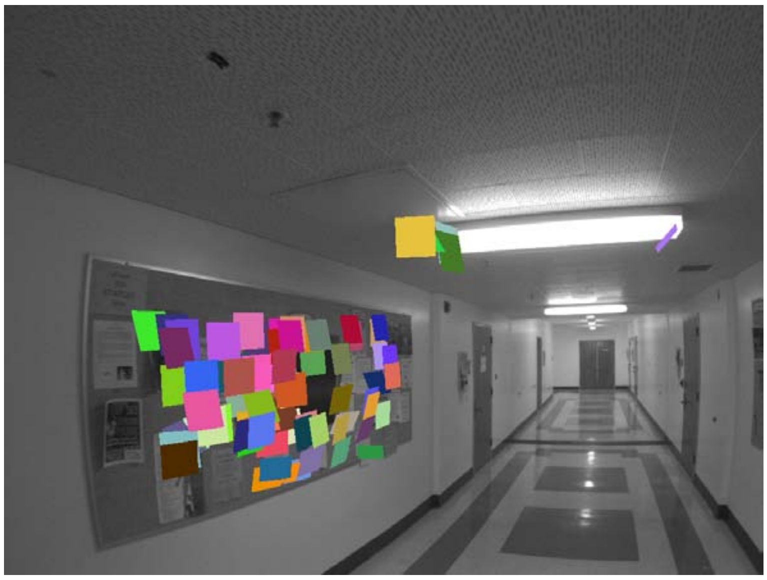


Figure 28: Example of patches back-projected in the reference frame.

A SURF descriptor is computed for each extracted patch by setting the key-point in the center of each patch and with parameters according to what said before.

As you will see in the results section and in fig. 30 all these computation leads to a nice results but some false positive loop closures still

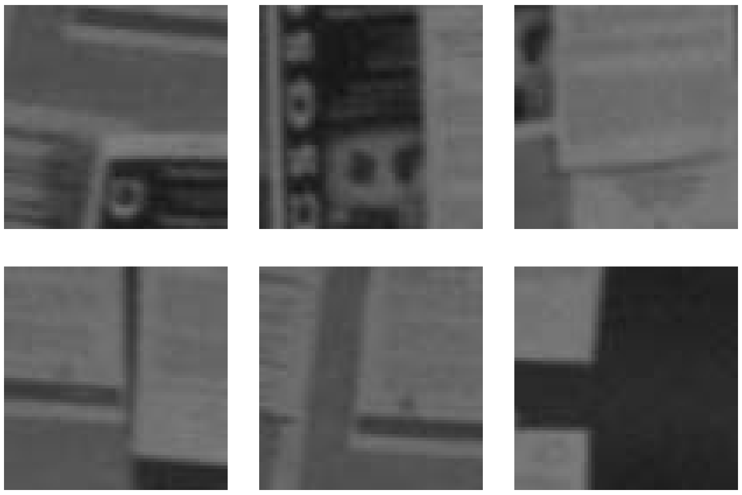


Figure 29: Examples of extracted patches.

being detected. To drop these false positives a geometric consistency metric on the spatial distribution of matched features is evaluated.

Only the subset M of features that has been matched in the comparison between the current frame and the candidate of the loop closure, is considered. Since the triangulation give us the estimation of the 3D points of the features, it is possible to compute all inter-features distances.

The new score metric have to consider both the number of matches and the similarity of the spatial distribution of features.

If $|M| = n$ we have $\binom{n}{2}$ couples of features and relative distances to evaluate.

A couple of feature (f_i^{cu}, f_j^{cu}) in the *current frame* is the counterpart of the couple (f_k^{ca}, f_l^{ca}) in the *candidate frame* if the features f_i^{cu} and f_j^{cu} has been respectively matched with the features f_k^{ca} and f_l^{ca}. A pair of such couples is called *related*.

Now consider for notation simplicity the symbol f_h^{frame} as both the feature and the 3D point related to that feature. The context can distinguish the usage.

The difference of the distances $\Delta d_t = |(\|f_i^{cu} - f_j^{cu}\| - \|f_k^{ca} - f_l^{ca}\|)|$ from each pair of related couple of features is computed.

Each Δd_t term have to contribute to the score by passing it to a function $\phi : \mathbb{R}^+ \rightarrow [0, 1]$:

$$\phi(\Delta d_t) = \begin{cases} 1 & \text{if } \Delta d_t = 0 \\ k & k \in [0, 1) \text{ otherwise} \end{cases} \tag{39}$$

To avoid discontinuity the function should continuously decrement its value as Δd_t increase. A suitable function is the exponential:

$$\phi(\Delta d_t) = e^{-k_{slope} \cdot \Delta d_t} \tag{40}$$

The variable $k_{slope} \in \mathbb{R}^+ \setminus \{0\}$ is an input parameter which determine how *fast* the function goes to 0.

The new score function become:

$$score = n \cdot \frac{\sum_{t=1}^{\binom{n}{2}} \phi(\Delta d_t)}{\binom{n}{2}} \tag{41}$$

The normalization by the number of couples and then the multiplication by the number of matches let us to obtain a score function that is sensible to both the spatial distribution of features and the cardinality of the matches set. The computation is made accordingly to the algorithm 6.

Algorithm 6: The score computation algorithm

```
   Input: $M, F^{cu}, F^{ca}$
   Output: The score $s$
   // Each element in $M$ is a pair of indices: the first
      relative to the current frame's feature set, the
      second to the candidate's features set
 1 $sum = 0$
 2 for $i$=1 to $n$-1 do
 3     for $j$=$i$+1 to $n$ do
 4         $d_1 = ||F^{cu}_{M[i].first} - F^{cu}_{M[j].first}||$
 5         $d_2 = ||F^{ca}_{M[i].second} - F^{ca}_{M[j].second}||$
 6         $\Delta d = |d_1 - d_2|$
 7         $sum = sum + e^{-k_{slope} \cdot \Delta d}$
 8     end
 9 end
10 return $\frac{2 \cdot sum}{(n-1)}$
```

8.6 RESULTS AND COMPARISONS

The proposed approach without the geometric check is capable to detect a loop closure in the desired area of the path. In fig. 30 there are the compared results in the form of confusion matrices (same as in the previous chapter). The white spot in the bottom left is our desired loop closure detection, the corresponding screen-shot is in fig. 31. But as you can see there is also a false positive in the version without the geometric check. That white spot correspond the screen-shot in fig. 32. This false positive can be easily dropped with the geometric verification, in fact on right there is no more false positives.

The first thing that is visible in the comparison of fig.33 is the different density of pixels and relative comparisons. In fact the number of comparisons is drastically reduced because of the need of couples

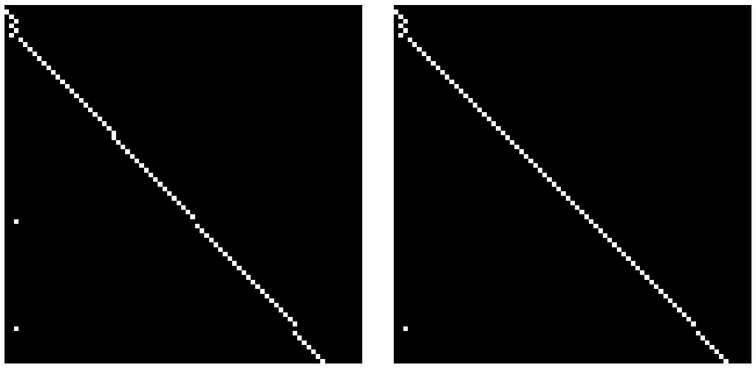
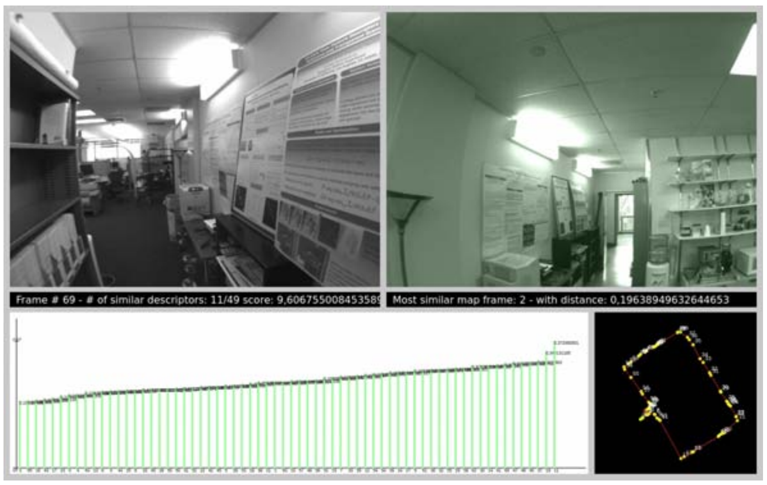


Figure 30: Results in the form of confusion matrix: on left the base approach with the false positive; on right with the geometric refinement to drop such false positive.



Frame # 69 - # of similar descriptors: 11/49 score: 9,60675500845358

Most similar map frame: 2 - with distance: 0,19638949632644653

Figure 31: The true positive loop closure detected by the proposed approach.

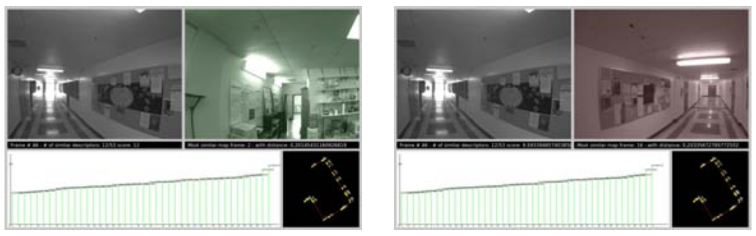


Figure 32: The false positive detected on left, rejected on right with the geometric verification.

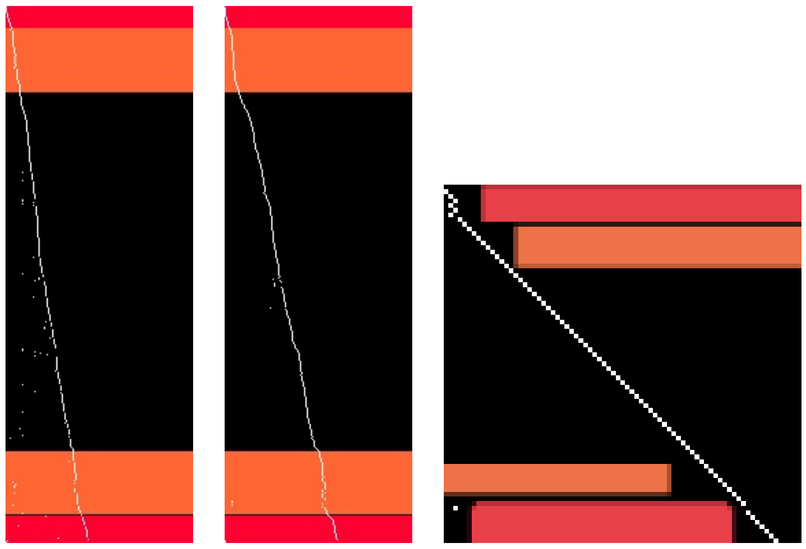


Figure 33: The of FAB-MAP and NV compared to the proposed approach.

of frames distant enough each other. This doesn't means that areas of the path are ignored, because such areas are the source of the triangulated features. As visible in fig. 34 the map is full covered except for a initial small place.

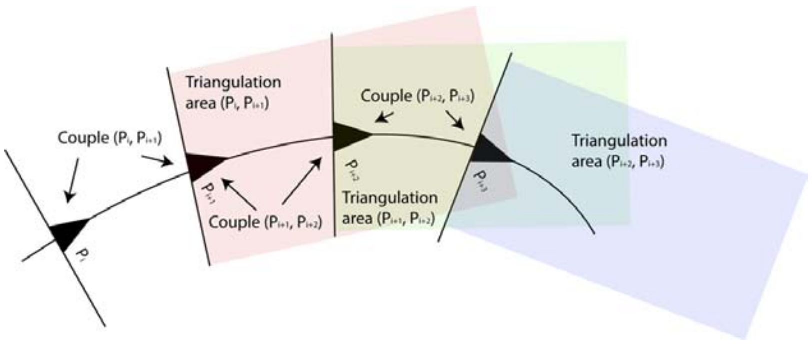

Couple (Pi, Pi+1)
Triangulation area (Pi, Pi+1)
Couple (Pi+2, Pi+3)
Triangulation area (Pi+2, Pi+3)
Couple (Pi+1, Pi+2)
Triangulation area (Pi+1, Pi+2)
Pi
Pi+1
Pi+2
Pi+3

Figure 34: Map coverage.

To make a comparison possible we coloured the corresponding parts of the corridor and the room in orange and red like for the FAB-MAP and NV confusion matrices.

As visible we dropped all false positive but at the price of no detection in the outer corridor. This is due to the triangulation that limit us to use only a portion of the image.

9

REMARKS

9.1 COMMENTS AND REMARKS

From the performance point of view almost all recent state of the art works run in real-time systems but with considerable differences in the scale that they support. Furthermore most works are not specific for humanoids and their problematics. Only a few number of approaches to solve SLAM problem for humanoids robots has been proposed during last decade. This kind of robots are expected to become our helpers in day life or at work, but lot has to be done before achieving that. Actual map representation are quite coarse to be used by humanoid robots to make high level tasks like tight a screw. Also humanoids itself have to be improved, in their control and planning. Such problems are very hard due to the high degree of freedom that humanoids offer.

Feature based SLAM demonstrated to be more efficient in term of computational costs. So future works may have to concentrate in this way.

The most natural approach that came up from state of the art works is the combination of an IMU and a video system [13] [30], because human effectively use them in a similar way. That system should be improved with some failure detection and recovery algorithm in order to help the robot if it falls or if it take a hit. This because the IMU can fail to record an acceleration and it make the estimated path to diverge from the ground truth.

An overall remark over these state of the art works is that the right way to develop an efficient and functional system is to take cutting-edge developed algorithm from other related research fields and try to apply them to the SLAM problem. A SLAM approach should be constructed in a modular way from the beginning so if a component from a different approach result more accurate or efficient it should be easily added to the system.

Most systems are divided in a back-end and a front-end ([13], [31], [20] and lot more). In the former take place all the optimizer that maintain the map given the constraint that arrive from the front-end. The front-end is responsible to the data association and in particular for the place recognition.

Restricting the view on the loop closure problem, the state of art works are divided in two main categories. One consider the loop closure detector as an oracle, and they put all their trust on it. Back-end based on optimization (like [15]) are very sensible to outliers and false positives and to avoid them a lot of computation is necessary.

In the other way of thinking the loop closure detector should be as light as possible. The light loop closure detector operate in the real-time front-end. A verification of the correctness is computed by the back-end [27]. This way the back-end should be elastic enough to support efficiently even severe modifications of the estimated path and map.

The proposed approach demonstrated the concept that the assumption of planar features in structured buildings can be used for a place recognition system for loop closures detections. Performances of the implementation was beyond the scope of this thesis, in fact it has bad timing performances due to the long optimization phase. It can take up to 1^h to elaborate the 10^m video used as input dataset. As future work a re-engineering process of the system should take place.

BIBLIOGRAPHY

[1] Pretto Alberto. *Visual-SLAM for Humanoid Robots*. PhD thesis, Universitá degli studi di Padova, Scuola di Dottorato di Ricerca in Ingegneria dell'Informazione, 2009.

[2] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory, vol. IT-14, no.*, May 1968.

[3] Mark Cummins and Paul Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *International Journal of Robotics Research*, 2008.

[4] Mark Cummins and Paul Newman. Appearance-only SLAM at large scale with FAB-MAP 2.0. *The International Journal of Robotics Research*, 2010.

[5] A.J. Davison, I.D. Reid, N.D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *Dept. of Comput., Imperial Coll., London*, June 2007.

[6] Ethan Eade and Tom Drummond. Unified loop closing and recovery for real time monocular slam. *In Proc. of the British Machine Video Conference, BMVC*, 2008.

[7] H.R. Evereth. *Sensors for Mobile Robot: Theory and Applications*. A.K. Peters LTD., 1995.

[8] Kanehiro Fumio et al. Whole body locomotion planning of humanoid robots based on a 3d grid map. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005.

[9] Dorian Galvez-Lopez and Juan D. Tardos. Real-time loop detection with bags of binary words. *IROS2011*, 2011.

[10] Giorgio Grisetti, Stachniss Cyrill, and Burgard Wolfram. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE TRANSACTIONS ON ROBOTICS, VOL. 23, NO. 1*, FEBRUARY 2007.

[11] S. Hinterstoisser, O. Kutter, N. Navab, P. Fua, and V. Lepetit. Real-time learning of accurate patch rectification. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[12] Gutmann Jens-Steffen, Fukuchi Masaki, and Fujita Masahiro. 3d perception and environment map generation for humanoid robot navigation. *Springer-Verlag Berlin and Heidelberg GmbH & Co. K*, September 2008.

[13] Eagle S. Jones and Stefano Soatto. Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *Submitted to the Intl. J. of Robotics Research, August 27, 2009 Revised May 10, 2010; Accepted September 23, 2010*, 2009.

[14] Sabe K., Fukuchi M., Gutmann J.-S., Ohashi T., Kawamoto K., and Yoshigahara T. Obstacle avoidance and path planning for humanoid robots using stereo vision. *Proceedings of the IEEE International Conference on Robotics and Automation*, 2004.

[15] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. on Robotics (TRO)*, 2008.

[16] Young Min Kim, Jennifer Dolson, Michael Sokolsky, Vladlen Koltun, and Sebastian Thrun. Interactive acquisition of residential floor plans. *ICRA, page 3055-3062. IEEE, (2012)*, 2012.

[17] Georg Klein and David Murray. Improving the agility of keyframe-based slam. *Computer Vision – ECCV 2008*, 2008.

[18] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

[19] YI Ma, Stefano Soatto, Jana Kosecka, and S. Shankar Sastry. *An Invitation to 3-D Vision*. Springer, 2004.

[20] J.B. McDonald, M. Kaess, C. Cadena, J. Neira, and J.J. Leonard. Real-time 6-dof multi-session visual slam over large scale environments. *Journal of Robotics and Autonomous Systems, RAS*, 2013.

[21] Montemerlo Michael and Thrun Sebastian. Fastslam: A scalable method for the simultaneous localization and mapping problem in robotics. *The International Journal of Robotics Research 2008 27: 1117*, January 2007.

[22] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 2005.

[23] Kwak Nosan, Stasse Olivier, Foissotte Torea, and Kazuhito Yokoi. 3d grid and particle based slam for a humanoid robot. *9th IEEE-RAS International Conference on Humanoid Robots*, December 2009.

[24] Stasse Olivier, J. Davison Andrew, Sellaouti Ramzi, and Yokoi Kazuhito. Real-time 3d slam for humanoid robot considering pattern generator information. *JRL, CNRS/AIST, ISRI*, June 2006.

[25] Oßwald S., Hornung A., and M. Bennewitz. Improved proposals for highly accurate localization using range and vision data. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, to appear.

[26] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots II edition*. The MIT Press, 2011.

[27] Niko Sunderhauf and Peter Protzel. Brief-gist - closing the loop by simple mean. *In proc. of the IEEE Intelligent Robots and System (IROS)*, 2011.

[28] Y. Takaoka, Y. Kida, S. Kagami, H. Mizoguchi, and T. Kanade. 3d map building for a humanoid robot by using visual odometry. *Proceedings of 2004 IEEE International Conference on Systems, Man and Cybernetics*, 2004.

[29] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623.

[30] Konstantine Tsotsos, Alberto Pretto, and Stefano Soatto. Visual-inertial ego-motion estimation for humanoid platforms. 2012.

[31] Thomas Whelans, Michael Kaess, John J. Leonard, and John McDonald. Deformation-based loop closure for large scale dense rgb-d slam. *IROS2013*, 2013.

[32] Jun Yang, Yu-Gang Jiang, Alexander G. Hauptmann, and Chong-Wah Ngo. Evaluating bag-of-visual-words representations in scene classification. *ACM Multimedia Information Retrieval (MIR)*, 2007.

[33] JUSTIN ZOBEL and ALISTAIR MOFFAT. Inverted files for text search engines. *ACM Computing Surveys*, 2006.