Weak Constraints Network Optimiser

Cyrille Berger

Abstract— **We present a general framework to estimate the parameters of both a robot and landmarks in 3D. It relies on the use of a stochastic gradient descent method for the optimisation of the nodes in a graph of weak constraints where the landmarks and robot poses are the nodes. Then a belief propagation method combined with covariance intersection is used to estimate the uncertainties of the nodes.**

The first part of the article describes what is needed to define a constraint and a node models, how those models are used to update the parameters and the uncertainties of the nodes. The second part present the models used for robot poses and interest points, as well as simulation results.

I. INTRODUCTION

A. Simultaneous Localisation and Mapping

Accurate localisation and accurate environment models are essential for a robot to accomplish its mission. While accurate localisation can be obtained from an accurate environment model, and the other way around, it is very seldom that accurate models or accurate localisation is available. It is therefore needed to compute both at the same the time, which is referred as the SLAM (Simultaneous Localisation and Mapping) problem.

An extensive overview of existing solution to this problem is provided in [4], [1]. The most famous and widely used solution to the estimation of landmarks positions and robot pose relies on the use of an Kalman filter [13]. However, since the SLAM problem is non-linear, a linearisation approximation is needed to be able to use a Kalman filter, also, in a real time system, the computational complexity of the algorithm limit the number of features that can be inserted in the map, as well as the memory requirement.

The main solution around this problem is the use of submap [5]: the robot construct independent local maps, with a limited number of features, and then a transformation between each submaps is recorded in a graph. When a loop closure happens, the position of each submaps is updated. However, the main drawback is that since maps are independent, when a landmark is shared between two maps, its parameters in one map are not updated with the observations made in the second map, which leads to poorly estimated parameters for those landmarks.

In [11], Piniès proposed a method that allow to share consistently some information between maps while keeping the submaps independent. The cost is an overhead on the size of each maps.

This work is supported by CUGS (Swedish National Graduate School in Computer Science)

Cyrille Berger is with the Department of Computer and Information Science, University of Linkping,SE-581 83 LINKPING, Sweden firstname.name@liu.se

An alternative to the use of the Kalman filter has also been investigated, most notably, the Graphical SLAM approach [6], [15], it is deriving from the EKF, a graph of robot pose and nodes is built, but it suffers from problems during loop closure. They are also limited to a robot evolving in a plane.

Those methods have the advantage to recover the full trajectory of the robot, and many of the improvement of those methods only estimate the trajectory, landmarks positions are to be estimated separately. The efficiency problem of those methods have been addressed by Olson in [10]. He proposed to insert robot pose into a tree, and then a rotation and translation error is computed for each constraint, and used to adjust the parameters of the pose along the tree.

Olson methods is limited to the second dimension, because it relies on the commutativity of rotations, which is only correct in two dimensions, in three dimensions, only rotation with the same axis can commute. But in [7], Grisetti demonstrated how to solve this problem, using a spherical linear interpolation [2], and his algorithm is capable of optimising a full trajectory in three dimensions, using full constraints between the nodes. However, their are two limitations for both algorithms, they are limited to the estimation of the trajectory, and they do not provide uncertainties for the robot poses. This algorithm has been generalised in [8] for any kind of graph of full constraints, but dropping support for incremental optimisation.

While it is possible to recover the landmarks positions given an accurate robot trajectory, we believe that the lack of estimation of landmarks parameters is a serious limitation of those algorithms. Especially since it implies the need for computing full constraint from each observation of the environment. This is acceptable in a two dimensions environment, with the use of a 2D scan laser, but in a 3D environment, this impose severe restriction on what kind of sensor can be used: either a stereo-vision system, or a 3D laser. Both are complex systems, relatively expensive, heavy and space consuming, which prevent to use them on many robotic systems, for instance small UAV. Also, even for stereo-vision based SLAM, it has been demonstrated that it is better to consider both camera as two different sensors [14], as it allows the modelling of uncertainties in the calibration.

The goal of this article is to address those problems, and our main contribution is to extend the optimisation algorithm presented in [7] to work with any kind of landmarks in three dimensions, as well as any kind of constraint with incremental updates. The second contribution is to provide a method to compute uncertainties. To achieve those goals a stochastic gradient descent method is used to update the parameters of the nodes in the graph, and belief propagation

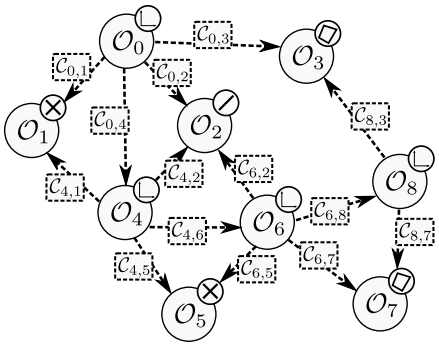


Fig. 1. Represents a graph of objects and the constraints between the objects, where $\mathcal{O}_0$, $\mathcal{O}_4$, $\mathcal{O}_6$ and $\mathcal{O}_8$ are robot poses, $\mathcal{O}_1$ and $\mathcal{O}_5$ are points and $\mathcal{O}_3$ and $\mathcal{O}_7$ are facets. $\mathcal{C}_{0,4}$, $\mathcal{C}_{4,6}$ and $\mathcal{C}_{6,8}$ are constraints coming from an odometer.

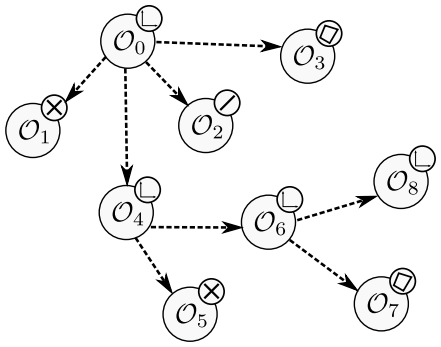


Fig. 2. Objects tree associated with the graph of figure 1. The connection represent the parent to child relation.

combined with covariance matrix intersection is used [16].

B. Graph modelling

When representing the SLAM problem as a graph, nodes represents objects $\mathcal{O}_i$, whose parameters are unknown, like robot poses and landmarks. And the edges are the constraints $\mathcal{C}_{i,j}$ between two objects $\mathcal{O}_i$ and $\mathcal{O}_j$. Those constraints are computed from observing the environment. For simplicity, objects are indexed in order of observation, so that if $i < j$, then object $\mathcal{O}_i$ is more ancient than $\mathcal{O}_j$.

Figure 1 shows an example of graph, with the constraints between objects.

The constraint network optimisation technique presented in [7] is limited to objects parametrised by a rotation and translation, which is the case of robots pose, and to a lesser extent to planes, but this does not work for other features like points or segments. Also in [7] the constraints have to contain the full transformation: rotation and translation.

For the algorithm presented in this article, any type of object parameters can be used, provided that a model of the objects is given, including the following functions:

- *initialisation* this function uses one or several constraints to compute a first estimation of the parameters of an object, as well as the associated uncertainty
- *rotation update* this function rotates the object on itself, in case the object is a point, this rotation can be ignored.
- *translation update* this function translate the object

Also, for each type of constraints, a different model of constraint will be used, and it will need to provide the following functions:

- *rotation error* $\hat{r}^c_{i,j}$ this function compute the rotation error between the two nodes, it is defined as the rotation that need to be applied on $\mathcal{O}_i$ so that the constraint is fully satisfied
- *translation error* $\hat{t}^c_{i,j}$. this function compute the translation error between the two nodes, it is defined as the translation that need to be applied on $\mathcal{O}_i$ so that the constraint is fully satisfied
- *constraint observation model* $h(\mathcal{O}_i, \mathcal{O}_j)$ it is the function that given the state of $\mathcal{O}_i$ and $\mathcal{O}_j$ computes the constraint parameters, the Jacobian of this functions are used to update the uncertainty of the state of each node.

The next section explains how this object and constraint models are used to compute an update of the parameters and of the associated uncertainties of the objects contained in the graph.

II. Weak Constraints Network Optimiser

A tree structure is used along the graph to keep the parameters of the graph, so that they are expressed locally with respect to a parent. One of the reason is that the global parameter of the objects is rather meaningless, in the sense that when the robot is far away from its initial position, the uncertainty on the position of the objects will be important. By using local information, the relative uncertainty with the surrounding object remains small. An other reason for the tree structure is that it will be used to propagate the rotation and translation adjustment between nodes of the graph. Also it is interesting to note that cutting a part of the tree would give us a local map of a part of the environment.

Figure 2 shows an example of tree associated to the graph presented in figure 1.

As in [10], [7], for each constraint $\mathcal{C}_{i,j}$ between the objects $\mathcal{O}_i$ and $\mathcal{O}_j$, given that $\mathcal{O}^A_{i,j}$ is the common ancestor of both objects in the tree, the optimisation process is going to minimise the rotation and translation error on the path $\mathcal{P}_{i,j} = \mathcal{O}_i \rightarrow \mathcal{O}^A_{i,j} \longrightarrow \mathcal{O}_j$ closed by the constraint $\mathcal{C}_{i,j}$.

The optimisation algorithm follows those steps:

1) Initialise the tree structure and the objects initial parameters
2) For each constraints $\mathcal{C}_{i,j}$
   a) Compute the rotation error $\hat{r}^c_{i,j}$ and use it to update the nodes parameters
   b) Compute the translation error $\hat{t}^c_{i,j}$ and use it to update the nodes parameters
3) For each node, update the uncertainty matrix
4) Repeat step 2 and 3 until the parameters have converged

The remaining of the section will go over the detail for each of the steps, first an explanation of how the tree structure is constructed is provided, then how the computation error and translation errors are propagated. Then the algorithm to

provide update for the uncertainty matrix is given, and the last part is a discussion on how to use the algorithm when building the map incrementally.

A. Tree structure

The tree structure is constructed in the same way as [7]. However, it is important to note that not all type of objects can be parents. Indeed, it is required that the parent is a pose, otherwise it would not be possible to express the parameters of the children in function of its parent: a robot pose is a pose, but points or segments are not, under some circumstances, a plane can be used as a pose [3]. Also since the tree is used to generate path for the update process, it is important to keep a good balance on the length of each path. A long path would allow to propagate the error correction quickly, but will have a high computational cost, while a short path will require more iteration to propagate the error.

A good compromise can be achieved by trying to minimise the length $l(\mathcal{O}_i)$ of the path between an object $\mathcal{O}_i$ and all the object it is connected to. Therefore the tree is constructed following those rules:

- The root of the tree is the first pose of the robot
- For an object $\mathcal{O}_i$ connected to a set of objects $\{\mathcal{O}_j\}$, the parent will be $\mathcal{O}_k$ the connected object with the shortest connection to the root:
  $\mathcal{O}_k \in \{\mathcal{O}_j\}$ such as: $\forall j, l(\mathcal{O}_k) < l(\mathcal{O}_j)$. In case $\mathcal{O}_k$ is not a pose, the parent of $\mathcal{O}_i$ will be set to be the parent of $\mathcal{O}_k$, as long as $\mathcal{O}_k$ was not observed at the previous time.

It shall be noted that constructing the tree in such a way does not guarantee to get an optimal tree, but it is efficient, and in case of a loop closure, it will still be able to ensure that the distance between the node of the two ends of the loop is minimal.

This is an other difference with the tree used by Grisetti in [7], since in our algorithm, it is not necessary true that there is a constraint between the parent and a child in the tree.

The reason that the parent of a landmark is used, instead of restricting to the objects directly connected, is that it would lead to the tree being constructed as a long chain of robot pose, and when the robot closes a loop the full chain would have to be evaluated, leading to a slower convergence rate.

B. Update objects parameters

As in [7], the rotation and translation errors are distributed on the objects of the path, and the update is done in two steps, first the rotation is updated, then the translation.

The path $\mathcal{P}_{i,j} = \left\{\mathcal{O}_l / l = p_k^{i,j}\right\}$ is the sequence of n objects that needs to be traversed in the tree to connect object $\mathcal{O}_i$ and $\mathcal{O}_j$. $p_k^{i,j}$ is suite that gives the indexes of the objects of the path, it is defined such as $p_1^{i,j} = i$ and $p_n^{i,j} = j$, and the index $R = p_r^{i,j}$ is the root of the path, such as $\mathcal{O}_R$ is the parent of $\mathcal{O}_Q$ and of $\mathcal{O}_S$ with $Q = p_{r-1}^{i,j}$ and $S = p_{r+1}^{i,j}$. $\mathcal{T}_{A\to k}$ is the transformation between the common ancestor and an object k in the path.

a) Update of the rotation: The rotation error is computed using the constraint model function $\hat{r}_{i,j}^c$, and its update is distributed along the path, using the slerp method [2]. In practise, if $\hat{r}_{i,j}^c = (\theta_{i,j}, \mathbf{x}_{i,j})$, where $\theta_{i,j}$ is the angle of the rotation and $\mathbf{x}_{i,j}$ its axis.

$$0 < k < r \quad r_k^{update} = \mathcal{T}_{i\to k} \otimes -r'_k \otimes \mathcal{T}_{i\to k}^{-1} \tag{1}$$

$$r < k \leq n \quad r_k^{update} = \mathcal{T}_{i\to k} \otimes r'_k \otimes \mathcal{T}_{i\to k}^{-1} \tag{2}$$

$$c_k = \lambda_r \cdot \frac{w_k}{\sum_{i=1, i\neq r}^{n} w_i} \tag{3}$$

$$r'_k = (c_k \cdot \theta_{i,j}, \mathbf{x}_{i,j}) \tag{4}$$

The node model is then used to update the parameters of the object k with the rotation r_k^{update}.

b) Update of the translation: The translation update is more straightforward, and it is done by simply translating the objects:

$$0 < k < r \quad t_k^{update} = -c_k \cdot \mathcal{T}_{A\to k} \cdot \hat{t}_{i,j}^c \tag{5}$$

$$r < k \leq n \quad t_k^{update} = c_k \cdot \mathcal{T}_{A\to k} \cdot \hat{t}_{i,j}^c \tag{6}$$

$$c_k = \lambda_t \cdot \frac{w_k}{\sum_{i=1, i\neq r}^{n} w_i} \tag{7}$$

The node model is then used to update the parameters of the object k with the translation t_k^{update}.

c) Damping coefficients: λ_r and λ_t are damping values, that are inferior to 1, to avoid the optimisation to overshoot and whose value is decreased iterations after iterations. It is also possible to give a higher value to λ_r and λ_t for constraints with the lowest uncertainty, this will gives the constraints with highest information more importance than the one with lowest.

While the w_k are computed from the uncertainty of the nodes parameters, a higher coefficient is given for nodes with higher uncertainty so that they get more updated than nodes whose parameters are already well known.

C. Update information matrix of objects

Since the steepest gradient descent does not provide a method to update the covariance matrix of the estimate of landmarks parameters nor of robot poses, it is necessary to use a different method. In [16] proposed an improvement to the loopy belief propagation [12] that does apply to pose graph (such as [7]), however this method is not directly applicable to our problem, since it computes a global uncertainty, and we are interested in computing local uncertainty. Also in [16], the spanning tree used to compute the uncertainty require the existence of a constraint between the parent and the child, while this is reasonable for a pose graph, in our case, when a loop closure happen, recent poses get connected to older poses in the tree, without the existence of a constraint between them.

Compared to [16], a more general formulation is used, given x_i the state of the node i, while $h_k(x_{i_k}, x_{j_k})$ is the

observation function of the constraint between node i_k and node j_k, z_k is the observation of the constraint. The goal of the algorithm is then to minimise the following function:

$$\delta\Theta^* = \underset{\delta\Theta}{\operatorname{argmin}} \sum_{k=1}^{K} \| H_{i_k} \delta x_{i_k} + J_{j_k} \delta x_{j_k} - c_k \|^2_{R_k} \quad (8)$$

Where R_k is the covariance of the zero-mean measurement noise, H_{i_k} and J_{j_k} are respectively the Jacobian of h_k with respect to a change in x_{i_k} and in x_{j_k}. c_k is the measurement prediction error: $c_k \triangleq z_k - h_k(x_{i_k}, x_{j_k})$.

The distributions parameters are then given by:

$$\Lambda_{ij} \triangleq \begin{bmatrix} \Lambda^{ii}_{ij} & \Lambda^{ij}_{ij} \\ \Lambda^{ji}_{ij} & \Lambda^{jj}_{ij} \end{bmatrix} = \begin{bmatrix} H^T_{i_k} \\ J^T_{j_k} \end{bmatrix} R_k^{-1} \begin{bmatrix} H^T_{i_k} \\ J^T_{j_k} \end{bmatrix}^T \quad (9)$$

For each constraint, the mutual information shared between the nodes i and j can be computed:

$$M^i_{ij} = \Lambda^{ii}_{ij} - \Lambda^{ij}_{ij} (\Lambda^{jj}_{ij} + M_j)^{-1} \Lambda^{ji}_{ij} \quad (10)$$

The resulting information matrix M^i is given by the intersection of all the mutual information M^i_{ji}, for the nodes which share a constraint:

$$M^i = \sum_j \omega_{ij} M^i_{ij} \quad (11)$$

Where $\sum_j \omega_{ij} = 1$, and ω_{ij} are chosen to maximize the determinant of M_i using a gradient descent technique.

D. Incremental updates

For an use in a robotic system, it is often desirable to get incremental update, so that the robot get benefits from better localisation and better environment model while performing its current task. The algorithm can be used in such a context rather easily: new poses, new landmarks and new observations are continuously added to the graph, and the iteration process can run in the background.

Also, it is good to reset the damping coefficients when new nodes are added, otherwise either old constraints gets more importance than new one, or the other way around.

III. Feature Models

The feature models are used to update the parameters of the features. Those update are rather straightforward since it is mostly a matter of a applying a rotation and a translation.

A. Pose

A pose is modelled as a rotation part R and as a translation t. The update is simply done by the following equations:

$$R_n = R_{up} \cdot R_{n-1} \quad (12)$$

$$t_n = t_{up} + t_{n-1} \quad (13)$$

The initialisation is done using the odometric information.

B. Point 3D

A point 3D is modelled with a translation t. The update is simply done by the following equation:

$$t_n = t_{up} + t_{n-1} \quad (14)$$

The initialisation process depends on whether a full observation of the point is available, like with a stereovision process, in which case the translation t is computed from the observation.

In case of a monovision process, only the direction d is known, and $t = \alpha \cdot d$, where α is not available from the observation. For the EKF filter, it has been suggested in [9] to use an inverse depth representation $t = d/\beta$, which allow to take an arbitrary β and associate it with a small covariance, which allow to keep the linearisation constraints of the EKF filter. However, an optimisation process is not sensitive to lineariation issues, but the convergence rate is faster if the initial values are close to the solution (it also limit the problem of local minima), it is therefore better to wait for a second observation and to compute the inital position as the line interesection.

C. Plane 3D

A plane in the environment does not give easy access to a pose, because while it is possible to define a normal and an origin, it is more tricky to measure accurately an orientation vector in the plane, we have define one in previous work [3], for small planes detected around interest points using a camera, but if the robot using a LIDAR it is not easily possible to define a stable orientation. However, not being able to define an orientation vector does not prevent to use the same model as the pose model, but it will be important to keep in mind that the orientation vector in the plane is meaningless, which has consequences when computing the rotation and translation error, but also it means that a plane cannot be used as the parent of other objects inside the tree.

IV. Constraints Models

The main purpose of the constraint models is to provide the rotation and translation error between the two nodes connected by the constraint. But the models are also used to compute update to the uncertainties.

In the context of the constraint models, the full path and graph does not matter, only the two object O_i and O_j as well as their common ancestor O_a in the tree.

A. Full 3D: between two poses

This model is used between two poses $B_i = (R_i, t_i)$ and $B_j = (R_j, t_j)$, expressed in their respective parent frame, while $(R_{a\to i}, t_{a\to i})$ and $(R_{a\to j}, t_{a\to j})$ are their parameters expressed in the common ancestor frame. The parameters of the constraint can be computed from odometers, GPS... It is perfectly acceptable to compute a constraint from two type of sensor, like a wheel and gyroscope.

The full 3D constraints includes a rotation part $R_{i\to j}$ and a translation part $t_{i\to j}$.

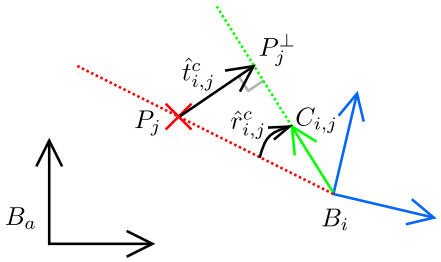


Fig. 3. Rotation and translation error for a point P_j observed from the pose B_i, with the constraint $C_{i,j}$.

d) Rotation error:

$$\hat{r}^c_{i,j} = R_i \cdot (R_{i\to j} \cdot R^{-1}_{a\to j} \cdot R_{a\to i}) \cdot R_i^{-1} \tag{15}$$

The term $\hat{r} = (R_{i\to j} \cdot R^{-1}_{a\to j} \cdot R_{a\to i})$ is the product of the constraint rotation by the parameters of both poses, this terms tend toward the identity rotation, until the constraint is fully satisfied.

$\hat{r}$ is then multiplied by R_i and R_i^{-1} because in three dimensions, rotations are not commutative. And the update equation for a node is given by equation 12, as you can see, in case the correction is only applied on the node i, using equation 1 to compute the value of the rotation update:

$$r_i^{update} = \hat{r}^c_{i,j} \tag{16}$$

$$R_i \leftarrow \hat{r}^c_{i,j} \cdot R_i = R_i \cdot \hat{r}^{-1} \tag{17}$$

Then if the new value of R_i is used in equation 15, the rotation error is equal to the identity.

e) Translation error: $\hat{t}^c_{i,j}$ is the translation given by the following transformation:

$$(R, \hat{t}^c_{i,j}) = \mathcal{T}_{a\to i} \otimes \mathcal{T}^c_{i\to j} \otimes \mathcal{T}^{-1}_{j\to a} \tag{18}$$

$$\hat{t}^c_{i,j} = R_{a\to i} \cdot t^c_{i\to j} + t_{a\to i} - t_{a\to j} \tag{19}$$

f) Constraint observation model: The constraint observation model is simply given by:

$$h_{i,j} = B_i^{-1} \otimes B_j \tag{20}$$

B. Direction to a 3D Point from a pose

This model is used between a pose $B_i = (R_i, t_i)$ and a point $P_j = (t_j)$, expressed in their respective parent frame, while $(R_{a\to i}, t_{a\to i})$ and $(t_{a\to j})$ are their parameters expressed in the common ancestor frame.

In this paper, we focus on the case where only the direction to the 3D point is known, as provided by a monocular system. Figure 3 shows the geometry associated with computed the rotation and translation error for this model.

g) Rotation error: It is simply define by the rotation around B_i that would move the point P_j on the line that follow the direction of the constraint $C_{i,j}$:

$$\hat{r}^c_{i,j} = R_i \cdot \left(cos^{-1}\left(\frac{\langle B_iP_j, C_{i,j}\rangle}{\|B_iP_j\|} \right), \frac{B_iP_j \wedge C_{i,j}}{\|B_iP_j\|} \right) \cdot R_i^{-1} \tag{21}$$

h) Translation error: In case of a stereovision model, it would be possible to use the constraint to compute the position of the point on the line of direction $C_{i,j}$, and from there the translation error would be given by the translation P_j to that point. But in the monovision case, the depth information is not available, and any point on the line would satisfy the constraint $C_{i,j}$, however it is preferable to minimise the translation, for the reason that it would minimise the error that will be created on the other constraints in the graph.

Given $P_j^{\perp}$ the orthogonal projection of P_j, the translation error is given by:

$$\hat{t}^c_{i,j} = P_j^{\perp} - P_j \tag{22}$$

i) Constraint observation model: The constraint observation model is simply given by:

$$h_{i,j} = SphericalAngles(R_i^{-1} \cdot (P_j - P_i)) \tag{23}$$

Where $SphericalAngles$ is a function that return the two spherical angles of a vector.

V. Simulation

We analyse simulation results for a large loop, a smaller loop but with an increase uncertainty, and finally a Monte Carlo simulation to evaluate the validity of the uncertainty model.

For the experiment, the following uncertainties models were used: the diagonal of the covariance for the odometry $(x, y, z, roll, pitch, yaw)$ is $(\sigma_t, \sigma_t, \sigma_t, \sigma_\theta, \sigma_\theta, \sigma_\theta)$. While the diagonal of the covariance for the direction vector between the robot and a point is $(\sigma_\omega, \sigma_\omega)$.

A. Large environment

In this experience, the robot is following a circular trajectory of 100m diameter, and its vertical position oscillate between $-1m$ and $1m$. The odometry has an accuracy of $\sigma_t = 0.01m/m$ for the translation parameters, and of $\sigma_\theta = 0.001rad/m$ for the angles parameters, while the accuracy on the direction for the interest points is $\sigma_\omega = 0.001rad$. We believe that those values are representative of an error model for average sensors.

In the environment of the simulation the points are set on two grids, at altitude $z = 0m$ and $z = 6m$ and the spacing of the grid is $6m$ (see figure 4 for images of the environment). And the sensor has a range of $10m$.

Figure 4 shows the difference between the estimation of the trajectory parameters and the reality. As expected, on the first part of the trajectory the optimisation process gives a result that is close to the odometry, until a loop closure is detected, at which point the optimisation process is able to give a much accurate position. As shown on figure 6, the uncertainty on the objects position is correctly propagated when a loop closure happen.

Figures 5 shows how the estimation of parameters for a few nodes evolves during the experiment. It is interesting to note that while the global error decreases during a loop

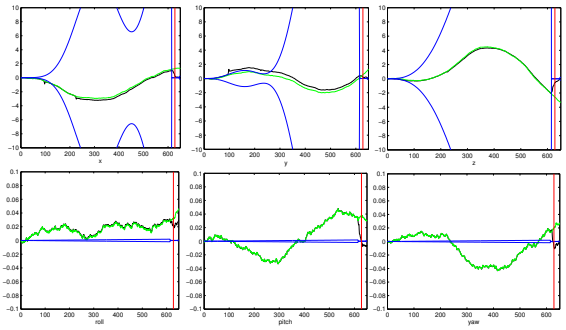


Fig. 4. Those figures show the evolution of the estimation of the position of the robot following the trajectory. The green curve is the error on position for the odometry, while the black curve is the error on the position estimated by the optimisation process, the blue curves are the uncertainties. The red line shows when the robot has reach the starting position.

closure, and the localisation of the robot is improved, the figure 5 shows that many individual nodes will get a less accurate position as a result of the optimisation. It is especially true for landmarks that were observed early on the trajectory.

As visible on figure 6, there is a delay on the update of the covariance during a loop closure, it is because the uncertainty is decreased when the tree structure is optimised, so that the parent of past pose node get a more direct connection to the root node. However, this optimisation of the tree structure takes several iterations.

B. Monte-carlo

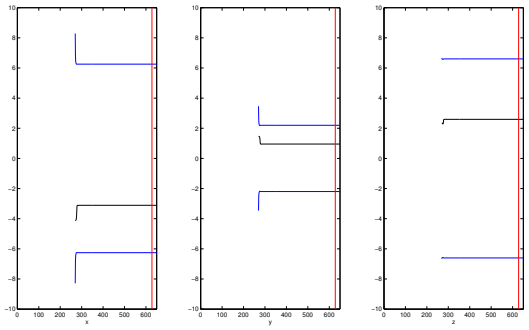
A Monte-carlo simulation with 50 runs was performed to evaluate how the uncertainty estimate performs. The normalised estimation error squared (NEES) of the current robot pose in the world frame was computed:

$$\epsilon_i = (x_i - \hat{x}_i)' \cdot P_i^- 1 \cdot (x_i - \hat{x}_i) \tag{24}$$

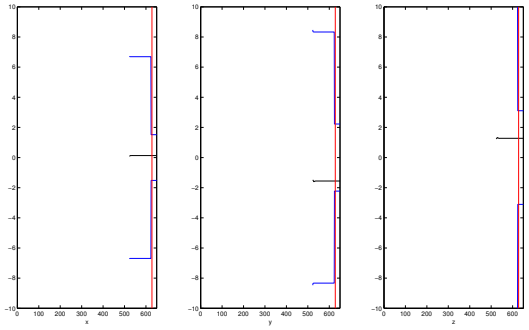
$$\bar{\epsilon} = \frac{1}{N} \sum_{i=1}^{n} \epsilon_i \tag{25}$$

For the simulation the robot is moving on a circle of radius of $40m$ of diameter, in the grid environment. The odometry has an accuracy of $0.01m/m$ for the translation parameters, and of $0.001rad/m$ for the angles parameters, while the accuracy on the direction for the interest points is $0.001rad$.

The result shown on figure 7 shows that globally the error is almost always consistent. The inconsistency happen during loop closure, which is not surprising, since the covariance intersection algorithm used to update the uncertainty of each node has a faster convergence rate than the steepest gradient descent algorithm used to update node position, which means it takes a few more steps before the consistency is restored. This also illustrate one of the problem of the method presented in this paper, since the covariance estimation and the parameters estimation are computed using two separate methods, there is no guarantee of consistency, even if it is generally achieved.



(a) point near the middle



(b) point near the end

Fig. 5. Those graphics show the evolution of the parameters estimation for different points in the graph. The blue curve is the estimated uncertainty, and the black curve is the error between the estimated parameters and the real parameters. The red line shows when the robot has reach the starting position.

C. Large uncertainty

In this experiment, the effect of increasing the uncertainty is studied to see how the optimisation process behaves. The robot is moving on a circular trajectory with a $40m$ diameter. The the following three settings are used ($\sigma_t = 0.01m/m, \sigma_\theta = 0.001rad/m, \sigma_\omega = 0.001rad$) (small uncertainty), ($\sigma_t = 0.01m/m, \sigma_\theta = 0.01rad/m, \sigma_\omega = 0.001rad$) (large uncertainty on the odometry) and ($\sigma_t = 0.01m/m, \sigma_\theta = 0.01rad/m, \sigma_\omega = 0.01rad$) (large uncertainty on both the odometry and the landmarks observation).

The results are shown on figure 8, they show that despite an important uncertainty, especially during loop closure, the optimisation process is capable of finding a good solution.

Also, the figure 8 shows that for the second time the robot run over the loop, the uncertainty get bigger than for the first time, this is because on the second and third run, the uncertainty on the robot position is mostly computed from the interest points, which provides limited information on the robot parameters. This effect is increased when the uncertainty on the landmark observations is increased.

VI. Conclusion and future work

A. Discussion

We have presented a SLAM technique that allow 3D mapping of the environment, using a graph of pose and features.

The main advantage of a graph optimisation technique does not lie in improvement on the position, but on the possibility to edit the graph, either to add new information or to fix errors. For instance, in a robotic system we can have sensor information coming from different sources, with a different rate, in a Kalman filter approach, all those data needs to be synchronised. With a graph approach if older data is processed after new data is added, it is still possible to add information to previous poses. Also, a wrong data association would break a Kalman filter, while in a graph approach it could be detected that the optimisation does not converge, and the faulty node could be found, and either removed or split. The information provided by the graph would have been wrong until the error is found, but then the graph will fix itself.

Also in previous work [17], we have shown how to model the environment using cooperative robots, with a multimap approach, one of the drawback of that approach is that when two robots have a rendez-vous they have to start a new map, which can lead to unfinished maps with poorly estimated features, also if two robots remains in view of each other for a long time, they would benefit of the other robot position only once, while with a graph approach a constraint can be added between both robot position each time they detect each other.

B. Future work

While our simulation results have shown that the algorithm works on the two most extreme case, when we have the most information for the node and the constraint (robot pose and full transformation), and when we have the least information (interest point and direction), it would be interesting to test the algorithm with a wider variety of input: global positions, stereovision, segments, planes...

During our experimentations, we have noticed that the damping factor used during the optimisation process yields a strong influence on both the quality of the results and the computational speed.

Also, keeping all the information is overkill, it would be beneficial to reduce the number of edges in the graph, and once the position of a node has been well observed, its edges should be concatenated into a single one.

References

[1] T. Bailey and H. Durrant-Whyte. Simultaneous Localisation and Mapping (SLAM): Part II - State of the Art. *Robotics and Automation Magazine*, September 2006.

[2] T. Barrera, A. Hast, , and E. Bengtsson. Incremental spherical linear interpolation. In *SIGRAD*, volume 13, pages 7–13, 2004.

[3] Cyrille Berger and Simon Lacroix. Using planar facets for stereovision slam. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

[4] H. Durrant-Whyte and T. Bailey. Simultaneous Localisation and Mapping (SLAM): Part I - The Essential Algorithms. *Robotics and Automation Magazine*, June 2006.

[5] C. Estrada, J. Neira, and J.D. Tardós. Hierarchical SLAM: Real-time accurate mapping of large environments. *IEEE Transactions On Robotics*, 21(4):588–596, August 2005.

[6] J. Folkesson and H. I. Christensen. Graphical SLAM for Outdoor Applications. *Journal of Field Robotics*, 24(1–2):51–70, February 2007.

[7] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Non-linear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems*, 10:428–439, 2009.

[8] Rainer Kuemmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A general framework for graph optimization. In *IEEE International Conference on Robotics and Automation*, 2011.

[9] J. M. M. Montiel, Javier Civera, and Andrew J. Davison. Unified inverse depth parametrization for monocular SLAM. In *RSS*, 2006.

[10] Edwin Olson, John Leonard, and Seth Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *IEEE International Conference on Robotics and Automation*, pages 2262–2269, 2006.

[11] Lina María Paz Pedro Piniés and J. D. Tardós. CI-Graph: An efficient approach for large scale SLAM. In *IEEE International Conference on Robotics and Automation*, pages 3913–3920, Kobe, Japan, May 12-17 2009.

[12] Ananth Ranganathan, Michael Kaess, and Frank Dellaert. Loopy SAM. In *Proceedings of the International Conference on Artificial Intelligence*, 2007.

[13] R. Smith, M. Self, and P. Cheeseman. A stochastic map for uncertain spatial relationships. In *Robotics Research: The Fourth International Symposium, Santa Cruz (USA)*, pages 468–474, 1987.

[14] Joan Solá, André Monin, Michel Devy, and Teresa Vidal-Calleja. Fusing monocular information in multi-camera SLAM. *IEEE Transactions on Robotics*, 2008.

[15] S. Thrun and M. Montemerl. The graphSLAM algorithm with applications to large-scale mapping of urban structures. *International Journal on Robotics Research*, 2005.

[16] Gian D. Tipaldi, Giorgio Grisetti, and Wolfram Burgard. Approximate covariance estimation in graphical approaches to SLAM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3460–3465, October 2007.

[17] Teresa Vidal, Cyrille Berger, and Simon Lacroix. Event-driven loop closure in multi-robot mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

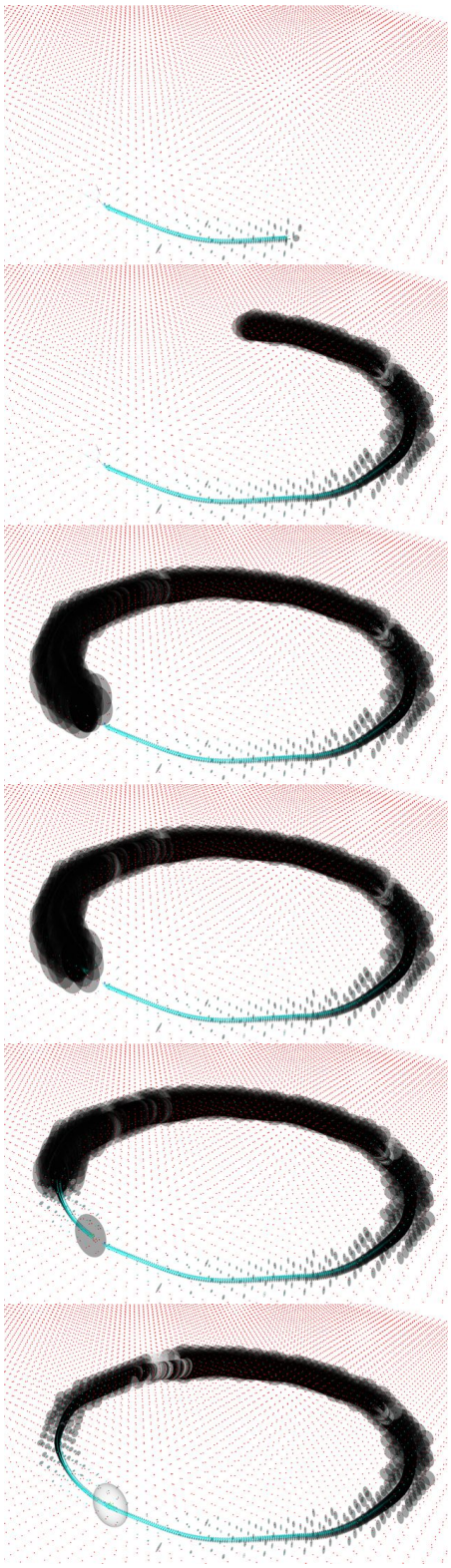


Fig. 6. Those figures shows the evolution of the environment model and the associated uncertainties, before and after the loop closure. The red dots are landmarks, cyan dots are features in the model. Robot poses are represented by the cyan arrows. And the black ellipsoid is the uncertainty associated with each object.

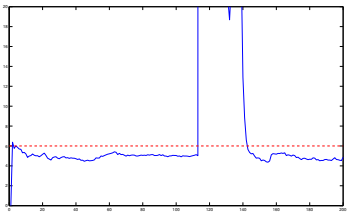
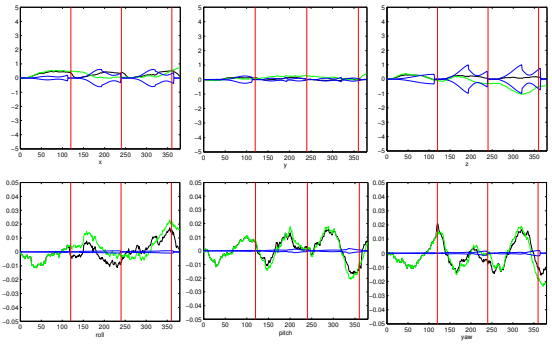
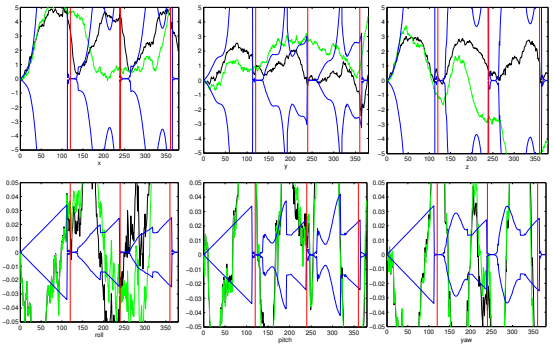


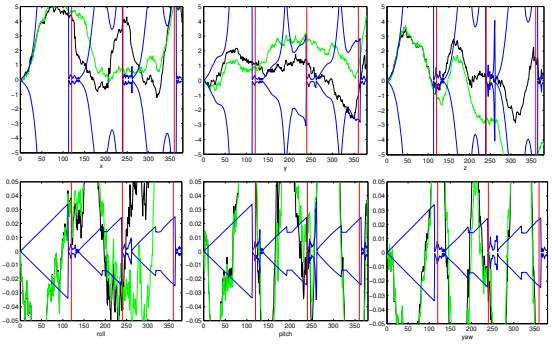
Fig. 7. Normalised estimation error squared of the current robot pose in the world frame, the red line shows the limit above which the value is estimated to be inconsistent.



(a) small uncertainty



(b) large uncertainty on the odometry



(c) large uncertainty on the odometry and the landmarks observations

Fig. 8. Those graphics show how the algorithms behave for larger uncertainty values. The blue curve is the estimated uncertainty, and the black curve is the error between the estimated parameters and the real parameters. The red line shows when the robot has reach the starting position.