Closed-form Online Pose-chain SLAM

Gijs Dubbelman and Brett Browning

***Abstract*— A novel closed-form solution for pose-graph SLAM is presented. It optimizes pose-graphs of particular structure called pose-chains by employing an extended version of trajectory bending. Our solution is designed as a back-end optimizer to be used within systems whose front-end performs state-of-the-art visual odometry and appearance based loop detection. The optimality conditions of our closed-form method and that of state-of-the-art iterative methods are discussed. The practical relevance of their theoretical differences is investigated by extensive experiments using simulated and real data. It is shown using 49 kilometers of challenging binocular data that the accuracy obtained by our closed-form solution is comparable to that of state-of-the-art iterative solutions while the time it needs to compute its solution is a factor 50 to 200 times lower. This makes our approach relevant to a broad range of applications and computational platforms.**

I. Introduction

We focus on the challenge of Simultaneous Localization and Mapping (SLAM) [17] and introduce a novel efficient and accurate closed-form solution called *Closed-form Online Pose-chain SLAM* (COP-SLAM). Improving on current SLAM solutions is relevant to many contemporary and future robotic applications such as intelligent vehicles, autonomous inspection and surveying platforms as well as supportive healthcare and domestic robots. The current consensus [21] is to use filter based solutions, e.g. EKF-SLAM, UKF-SLAM or FAST-SLAM [23], when computational resources are limited and to use more accurate graph based [9], [10], [12], [15], [19], [22], [24] or bundle adjustment based [11], [13] solutions when sufficient computational resources are available.

Systems using these more computationally demanding approaches typically consist of front-end and back-end subsystems. The responsibility of the front-end is to provide an initial solution, e.g. by using (visual) odometry [7], [18], [25] or structure-from-motion [3], [21] type of methods, as well as to detect loops in the robot's trajectory, e.g. by using appearance based loop detection [4], [16] or local map matching [2]. The responsibility of the back-end is to turn the initial solution into a maximum likelihood solution considering all available data. Such approaches are typically more accurate than filter based approaches as they are better able to capture the non-linearity that is inherent to SLAM.

A distinction between laser based and image based systems is that in the latter case the back-end optimizer typically provides the robot's pose at each time-step together with a sparse map of the robot's environment (in contrast to the more dense maps obtained by modern laser scanners). This sparse map is often turned into or replaced by a dense map using additional algorithms, e.g. PMVS [8], within a background (or offline) process. Alternatively, the actual map can be the collection of all images within the appearance database together with their absolute poses obtained by the SLAM back-end [1], [14]. The purpose of the sparse map in such settings is therefore more to provide a means to estimate absolute poses rather than providing the actual (dense) map itself.

When this is the case, one can choose to only optimize with respect to the absolute poses and not with respect to the sparse map within the SLAM back-end. Graph based techniques employing this are called pose-graph optimizers. They rely on the SLAM front-end to turn the edges (i.e. the observations) in the graph between landmarks and absolute poses into edges directly between the absolute poses themselves. These new edges are the relative pose displacements between absolute poses and the statistical information of all landmark observations is represented for by the uncertainty measures (e.g. covariance matrices) of all relative pose displacements. What we have gained is that, because there are typically less poses than landmarks, we are now solving a much smaller SLAM problem.

In this contribution we present our novel COP-SLAM solution to pose-graph SLAM. Its input are pose-graphs in which there are relatively few edges between nodes. Such sparse pose-graphs are called pose-chains and are described in Sec. II. Single loop pose-chains can be optimized directly using trajectory bending. The original trajectory bending algorithm of [6] is therefore briefly recapitulated in Sec. II-A. In order to use this technique on trajectories consisting of multiple loops, certain extensions to the original trajectory bending algorithm are required. These extensions are provided in Sec. III along with an algorithmic description of COP-SLAM. In Sec. IV we discuss the difference between the optimality conditions of COP-SLAM and that of existing non-linear iterative solutions. The practical relevance of these differences is explored with extensive experiments in Sec. V. These experiments focus on the domain of binocular visual-SLAM and a total of 49 kilometers of challenging trajectories are used for our experiments. Our conclusions are provided in Sec. VI.

II. From graphs to pose-chains

The conceptual differences between general graph-based SLAM, pose-graph SLAM and pose-chain SLAM is depicted in Fig. 1. A few decades ago the SLAM front-end typically

This report was made possible by the support of an NPRP grant from the Qatar National Research Fund. The statements made herein are solely the responsibility of the authors. Gijs Dubbelman is with Eindhoven University of Technology, g.dubbelman@tue.nl. Brett Browning is with CMU's Robotics Institute, brettb@cs.cmu.edu

consisted of regular odometry (wheel rotation and steering angle) with for example a single laser range scanner. Such front-ends produced relatively erroneous initial estimates creating the need to optimize with respect to poses and landmarks within the back-end (Fig. 1.a). As sensor technology and processing power advanced over the years the methods used inside front-ends started to produce ever more accurate initial estimates. This lowered the requirements given to the SLAM back-end which made pose-graph SLAM a popular technique (Fig. 1.b). Current state-of-the-art visual odometry and appearance based loop detection techniques have significantly increased in accuracy and reliability, and are tractable with modern computational hardware As the error drift of such modern techniques is relatively low, it is not always necessary to keep track of all edges between absolute poses in the SLAM front-end. We call the resulting sparse graph a pose-chain (Fig. 1.c).

The reduction of edges in the graph of pose-chain SLAM with respect to the graph of pose-graph SLAM is not the result of an explicit marginalization process. It is a direct consequence of the type of environment and observation processing being used. We will come back to this in our experimental section. The utility of pose chains was demonstrated in the work on trajectory bending [6] where the only links used were the relative pose displacements between successive absolute poses and one link related to a loop-detection of a large cyclic part of the robot's trajectory. The work on trajectory bending was limited to trajectories consisting of a single loop. Here we extend to multiple loops, as in Fig. 1.c. A key property of a pose-chain is that nodes are strictly ordered in time. While this is a very natural property for SLAM systems, there is no such restrictions for pose-graphs in general. Another property of pose-chains is that they have two classes of edges. The first class is that of *successive edges* which connect successive nodes, i.e. those from time step t to $t+1$. The second class is that of *loop-closing edges* which connect nodes from time step t back to time step $t-l$ for a relatively large l, e.g. $l > 100$. These edges are in some way special as they contain vital information on the drift of the initial estimate for the vehicle trajectory.

Our solution allows optimizing such pose-chains in closed-form and in linear time complexity wrt. the number of poses. It is therefore highly stable and efficient. At its core it performs an extended version of trajectory bending every time a loop is detected.

A. Trajectory bending

Conceptually speaking, trajectory bending updates each element in a chain of relative poses such that the final absolute pose is equal to a desired absolute pose (Fig. 2). From a fundamental perspective the chain can consist of transformations belonging to a Lie group whose exponential map and logarithmic mappings are computable for all its elements. The most important Lie groups for robotic purposes are the groups of Euclidean motions SE(2) and SE(3) and their rotational and translational subgroups SO(2), SO(3), R2 and R3. Here we briefly recapitulate the original trajectory bending algorithm of [6]. We describe it in general terms of Lie groups in order to facilitate our notation later on.

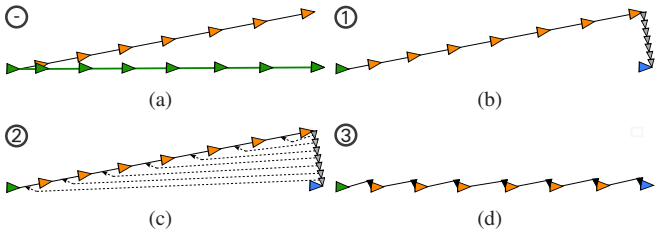


Fig. 2. Conceptual illustration of the original trajectory bending algorithm of [6]. In (a) the absolute poses of the ground truth trajectory are depicted by the green triangles and the ground truth relative pose displacements by the green edges. Estimates for the absolute poses of the trajectory are depicted by the orange triangles and estimates for the relative pose displacements by black edges. At a certain point in time the system obtains more accurate information on its last absolute pose, i.e. the blue triangle in (b). The first step is to find relative pose updates, shown by the small light gray triangles in (b), which bring the last absolute pose onto the desired last pose, i.e. the blue triangle. These updates are called the *local updates*. The second step is to compute transformations which distribute these local updates over the trajectory (c). The result of these transformations are the *distributed updates* which are depicted in (d) by the small black triangles. In (d) an improved trajectory is computed using the relative pose displacements together with the distributed updates. The result is that the trajectory ends in the desired absolute pose.

Fig. 3. Example images from the three binocular data sets used for our experiments.

Let $\mathrm{M}_1, \mathrm{M}_2, ..., \mathrm{M}_n$ be a set of *relative transformations* which all are elements of the Lie group $\mathcal{M}$ (e.g. the Lie group of rotation matrices in 3D). The elements of the set are strictly ordered in time and their subscripts denote their time steps. Each transformation also comes with its own uncertainty measure, these are expressed by the variances $\sigma_1^2, \sigma_2^2, ..., \sigma_n^2$.

The *absolute transformation* A_t at time step t can be computed with

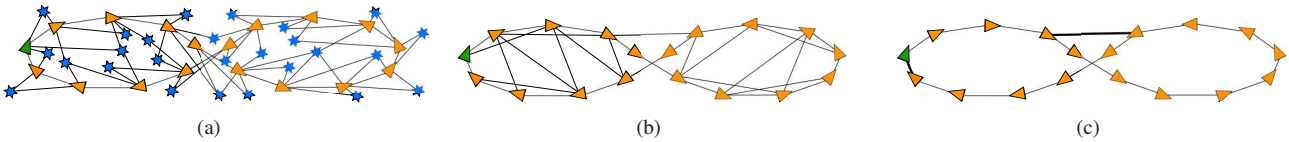
$$\mathrm{A}_t = \prod_{i=1}^{t} \mathrm{M}_i = \mathrm{M}_1 \star \mathrm{M}_2 \star \mathrm{M}_3 ... \star \mathrm{M}_t \qquad (1)$$

where $\star$ is the Lie group operator (e.g. matrix multiplication). As was illustrated in Fig. 2 the goal is to update each relative transformation such that the last absolute transformation A_n is exactly equal to a desired absolute transformation D_n.

The first step is to compute the n *local updates* $\hat{\mathrm{U}}_1, \hat{\mathrm{U}}_2, ..., \hat{\mathrm{U}}_n$ such that

$$\mathrm{D}_n = \mathrm{A}_n \star \prod_{i=1}^{n} \hat{\mathrm{U}}_i \ , \qquad (2)$$

i.e. when putting all local updates behind the last absolute transformation A_n the result is equal to the desired transformation D_n. This is illustrated in Fig. 2.b. Each local update



(a) (b) (c)

Fig. 1. Illustration of different types of graph based SLAM. General graph SLAM (a), pose-graph SLAM (b), pose-chain SLAM (c). The initial absolute pose is depicted by the green triangle and all successive absolute poses by orange triangles. Landmarks are shown by blue stars. Edges between nodes (i.e. absolute poses and landmarks) are depicted by black lines. In (a) edges model landmark observations in (b,c) they model relative pose displacements. In (c) the two loop-closing edges are marked by heavy black lines.

can be computed using the relative interpolation function

$$\hat{\mathrm{U}}_t = \mathfrak{I}(\sum_{i=1}^{t-1} w_i)^{-1}\mathfrak{I}(\sum_{i=1}^{t} w_i), \tag{3}$$

where

$$\mathfrak{I}(\alpha) = \mathrm{A}_n \star \exp(\alpha \log(\mathrm{A}_n^{-1} \star \mathrm{D}_n)). \tag{4}$$

Here $\exp$ and $\log$ denote the exponential and logarithmic mappings of the Lie group $\mathcal{M}$, e.g. see [20]. The normalized weights $w_1, w_2, ..., w_n$ are computed from the variances with

$$w_t = \frac{1/\sigma_t^2}{\sum_{i=1}^{n} 1/\sigma_i^2}. \tag{5}$$

They determine how much of the update is distributed to a particular transformation in the chain. The more uncertain a transformation is, the more it will be updated relatively to the other transformations.

The next step, illustrated in Fig. 2.c, is to distribute the local updates over all relative transformations. We thus seek a set of *distributed updates* $\mathrm{U}_1, \mathrm{U}_2, ..., \mathrm{U}_n$ such that

$$\mathrm{D}_n = \prod_{i=1}^{n}(\mathrm{M}_i \star \mathrm{U}_i) = \mathrm{A}_n \star \prod_{i=1}^{n} \hat{\mathrm{U}}_i \ . \tag{6}$$

Instead of putting all the updates behind the last absolute transformation A_n, each relative transformation M_i is succeeded by its own distributed update U_t. The relation between each local update $\hat{\mathrm{U}}_t$ and its distributed update U_i is given by

$$\mathrm{U}_t = \mathrm{A}_t^{-1} \star \mathrm{D}_n \star \hat{\mathrm{U}}_t \star \mathrm{D}_n^{-1} \star \mathrm{A}_t. \tag{7}$$

It involves nothing more than a change of basis.

The final step is simply to recompute all new absolute transformations given the relative transformations with their distributed updates. The result is that the last absolute transformation is exactly equal to D_n as illustrated in Fig. 2.d. All these steps can be computed in closed-form and in linear time complexity wrt. the number of poses.

III. COP-SLAM

In order to extend the original trajectory bending algorithm of Sec. II-A to a fully featured SLAM optimizer three extensions must be made:

The original trajectory bending algorithm will update the chain of relative transformations such that it ends exactly in the desired absolute transformation. By doing so it neglects the fact that desired transformation may be subject to uncertainty. We thus need a mechanism that takes into account the uncertainty of the chain of relative transformations and the uncertainty of the desired absolute transformation in an optimal way. This mechanism is described in Sec. III-A.

The most important shortcoming of the original trajectory bending algorithm is that it can only be used for transformation chains encompassing a single loop. The naive solution would be to run the original trajectory bending algorithm every time a loop is detected solely on that single loop. This however neglects the fact that the chain of transformations can have loops within loops and using this naive approach will not take into account the information of previous loop closures. In Sec. III-B we therefore provide an extension to the original trajectory bending approach that allows it to be used on multi-loop transformation chains.

In [5] it was proven that, under the assumption that the uncertainty of transformations are isotropic, trajectory bending provides an optimal solution only for Lie groups whose logarithmic map is related to a bi-invariant distance metric over the Riemannian manifold associated to the Lie group. Lie groups with this property are for example SO(2), SO(3), R2 and R3. Unfortunately this does not include SE(2) nor SE(3), which are the most important Lie groups to robotics. In Sec. III-C we develop an extension to the original trajectory bending algorithm which makes it piecewise optimal for SE(2) and SE(3).

An algorithmic description of COP-SLAM incorporating all these extension is provided in Algorithm 1.

Algorithm 1 COP-SLAM

while running **do**
  Get edge from SLAM front-end
  if edge is a *loop-closing edge* **then**
    Restrict computations to the Lie group of rotations.
    R.1) Get $\sigma^2_{\mathrm{D}_n}$ and compute $\sigma^2_{\mathrm{A}_n}$ with Eq. 9.
    R.2) Use Eq. 10, 3, 4 and 7 to compute the distributed updates.
    R.3) Multiply σ of each edge with the β of Eq. 13.
    Recompute all absolute poses given the distributed updates.
    Restrict computations to the Lie group of translations.
    T.1) Get $\sigma^2_{\mathrm{D}_n}$ and compute $\sigma^2_{\mathrm{A}_n}$ with Eq. 9.
    T.2) Use Eq. 10, 3, 4 and 7 to compute the distributed updates.
    T.3) Multiply σ of each edge with the β of Eq. 13.
    Recompute all absolute poses given the distributed updates.
  else
    Add *successive edge* to end of pose-chain and compute new last absolute pose
  end if
end while

A. Incorporating uncertainty

The original trajectory bending algorithm of [6] ignores the uncertainty $\sigma^2_{\mathrm{D}_n}$ of the desired transformation D_n. In [5] this was circumvented by first optimally fusing the last transformation A_n with the desired transformation D_n and then feeding the result of this fusion F_n to the trajectory bending algorithm. This is illustrated in Fig. 4.a. As we are dealing with general Lie groups this fused result F_n is computed with

$$\mathrm{F}_n = \mathrm{A}_n \star \exp\left(\frac{1/\sigma^2_{\mathrm{A}_n}}{1/\sigma^2_{\mathrm{A}_n} + 1/\sigma^2_{\mathrm{D}_n}} \log(\mathrm{A}_n^{-1} \star \mathrm{D}_n)\right). \quad (8)$$

where $\sigma^2_{\mathrm{D}_n}$ expresses the uncertainty in the desired last transformation and $\sigma^2_{\mathrm{A}_n}$ expresses the uncertainty in the last absolute transformation. When the Lie group's logarithmic map is related to a bi-invariant Riemannian distance metric over the Riemannian manifold associated to the Lie group, then the uncertainty σ_{A_n} can simply be computed with

$$\sigma^2_{\mathrm{A}_n} = \sum_{i=1}^{n} \sigma_i^2. \quad (9)$$

We now need a mechanism which performs this fusion within the trajectory bending algorithm itself. Note that the fused transformation F_n lays on the minimizing geodesic from A_n to D_n such that the ratio between its distance to A_n and its distance to D_n is $\sigma^2_{\mathrm{D}_n}/\sigma^2_{\mathrm{A}_n}$. The relative interpolation function of Eq. 4 also interpolates over the minimizing geodesic from A_n to D_n. All that is needed is to assure that the interpolation ends at F_n instead of at D_n. This is illustrated in Fig. 4.b and can be accomplished by changing the function of Eq. 10 that calculates the interpolation weights. It is straightforward to verify that when computing the interpolation weights according to

$$w_t = \frac{1/\sigma_t^2}{1/\sigma^2_{\mathrm{D}_n} + \sum_{i=1}^{n} 1/\sigma_i^2}, \quad (10)$$

the local updates end at F_n and hence after trajectory bending the last absolute transformation is equal to the optimally fused transformation F_n instead of D_n as desired.

For later purposes the uncertainty in F_n and thus also the uncertainty in the last transformation after trajectory bending A_n can be computed with

$$\sigma^2_{\mathrm{F}_n} = \frac{1}{1/\sigma^2_{\mathrm{A}_n} + 1/\sigma^2_{\mathrm{D}_n}}. \quad (11)$$

B. Multiple loops

In order to deal with transformation chains that encompass multiple loops, a mechanism is needed that respects the information of previous loops when a new loop is detected. We can accomplish this by updating the variances associated to each relative transformation.

The variances express how accurate a certain relative transformation is expected to be with respect to the other transformations. At the time of loop-closure the desired

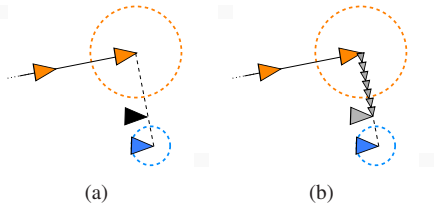


Fig. 4. External (a) and internal (b) fusion mechanisms for trajectory bending. The uncertainty in the last absolute pose is depicted by the orange circle and the uncertainty in the desired pose by the blue circle. The optimal combination of the last absolute pose and the desired pose is shown by the black triangle in (a). In (b) the local updates end at the large gray triangle which has the same position as the optimal combination of the last absolute pose and the desired pose.

absolute transformation adds important information to the chain making all the relative transformations within the loop more accurate. What we need is a mechanism to optimally compute new values for the variances of relative transformations within the loop such that their increase in accuracy is accounted for. The variance values of relative transformation that are not within the loop are not affected by this loop-closure information and keep their original values.

The effect is that when a future loop is closed that encompasses the previous loop as well transformations which were not part of the previous loop, then the interpolation weights of transformations inside the previous loop are lower than those of transformations not inside the previous loop. When closing this new loop, bending will therefore focus mostly on the less accurate transformations that were not inside the previous loop and less on the more accurate transformations that were part of the previous loop. This is only a conceptual description and we seek a formal and optimal mechanism to assign and update the variances of all relative transformations in a loop after it is closed.

Consider that before a loop is closed the uncertainty of the last absolute transformation is obtained by summing all relative transformation uncertainties with Eq. 9. After loop closure the chain ends in the optimal combination of the old last transformation and the desired transformation. The uncertainty in the last transformation after loop-closure is provided by Eq. 11. The goal is to update all relative transformation uncertainties such that when they are summed up in Eq. 9, the result is equal to the new uncertainty of the last transformation provided by Eq. 11. Put more formally, we seek a scalar β such that

$$\sum_{i=1}^{t} \beta\sigma_i^2 = \frac{1}{1/\sigma^2_{\mathrm{A}_n} + 1/\sigma_{\mathrm{D}_n}}, \quad (12)$$

i.e. when summing up all relative transformation uncertainties which are updated by multiplying with β, the result should be equal to the uncertainty of the new last transformation after loop-closure (which itself is the optimal combination of the old last transformation and the desired transformation). After some straightforward manipulation we

find that the optimal value for β is provided by

$$\beta = \frac{1}{1 + \sigma^2_{\mathrm{A}_n} / \sigma^2_{\mathrm{D}_n}} \tag{13}$$

.

C. Piecewise optimality for SE(2) and SE(3)

As pointed out earlier the original trajectory bending approach provides sub-optimal results for SE(2) and SE(3). Here we improve trajectory bending such that it provides a piecewise (i.e. subgroup) optimal solution for SE(2) and SE(3).

The solutions is as follows. At loop-detection we first apply trajectory bending solely to the subgroup of rotations, i.e. either SO(2) or SO(3). This will close-the-loop in the rotational subspace and will on average improve all relative rotations. Then we reintegrate the trajectory to obtain the improved absolute positions given the improved relative rotations. We then perform trajectory bending to the translational subgroup, i.e. either R2 or R3. The end results is that the loop is closed in both the rotational and translational subspaces. What we have gained is that both steps involve Lie groups for which trajectory bending computes an optimal solution. This approach is therefore piecewise optimal. A more detailed description is provided in Algorithm 1.

At first it may seem that this requires more computation than the original algorithm as we are applying trajectory bending twice. The opposite is true, it is more efficient. The number of floating point operations required when dealing with rotations and translation separately is less than when dealing with them within Euclidean motions. For example multiplying two elements of SE(3) will take 36 multiplications and 27 additions. Multiplying two element of SO(3) takes 27 multiplications and 18 additions and adding two translations in 3D takes 3 additions. So that is a total of only 27 multiplications and only 21 additions when dealing with rotations and translations separately. A similar reduction is observed for computing the inverse of elements. These reductions in floating point operations out weight the additional steps of our piecewise optimal algorithm.

IV. Maximum Likelihood solutions

The only known pose-chain optimizers obtaining maximum likelihood (ML) solutions for SE(2) and SE(3) under general conditions are non-linear and iterative in nature, examples are the methods of Gauss-Newton and Levenberg-Marquardt. Efficient implementations of such optimizers are at the core of SLAM back-ends such as G^{2}O [15], TORO [10] and [19]. COP-SLAM cannot obtain the same levels of optimality and there are two reasons for this.

The first reason is that at its core it performs trajectory bending which itself is only piecewise optimal for for SE(2) and SE(3) when edge uncertainties are isotropic. The underlying principle is that COP-SLAM cannot actively reduce errors in absolute positions by altering relative rotations as non-linear iterative methods can do. COP-SLAM first reduces absolute orientation errors by altering relative rotations. It then reintegrates the trajectory such that the absolute positions are improved given the on averagely improved relative rotations. This is however a passive process and COP-SLAM cannot make an active trade-off between errors in absolute positions and errors in absolute rotations.

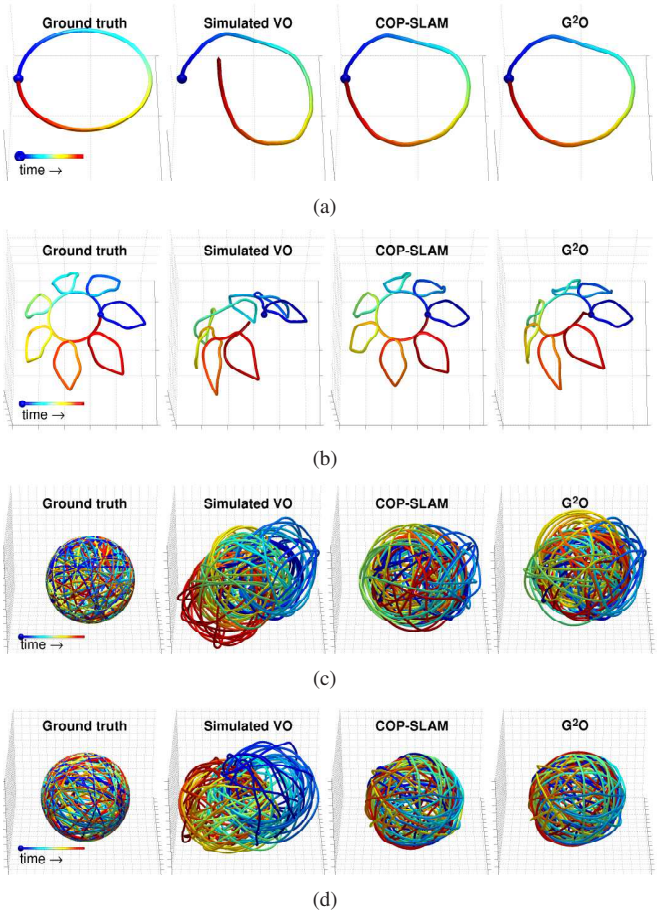


Fig. 5. Results of both methods obtained on 1 of the 100 simulated trajectories for each data set. Loop is shown in (a), flower in (b), world with 5 loops in (c) and world with 25 loops in (d). From left to right: ground truth, simulated visual odometry, COP-SLAM, G^{2}O.

The second reason is that COP-SLAM only optimizes one loop at a time. It will take into consideration information of previous loop closures but it will not back-propagate information of a new loop closure past the start of this new loop. When the new loop encompasses all previous loops or when the new loop is completely detached from all other loops (as in 5.a and 5.b) COP-SLAM's solution is still piecewise optimal. But the more loops are interconnected with each other, the more sub-optimal COP-SLAM's solution will be.

In defense of COP-SLAM's closed-form solution we can say that using optimal non-linear iterative methods comes with the risk of ending up in local minima as well as reduced stability while needing to spend more computations. The question therefore arises: in which practical circumstances do the advantages of non-linear iterative optimization outweigh its disadvantages with respect to the performance of COP-SLAM's closed-form solution.

V. EVALUATION

We investigate the performance difference between our closed-form COP-SLAM approach and methods providing ML solutions. In recent work [15] several methods were compared against the G^{2}O back-end optimizer. It was shown that G^{2}O provides either better or comparable accuracy and efficiency than alternative optimization methods. Therefore, here we only compare to G^{2}O.

During initial experiments we observed that the the Gauss-Newton method with the CHOLMOD solver of G^{2}O outperformed other solvers in accuracy and efficiency on our data sets, we therefore only report the performance using this optimizer and solver. The original online version of G^{2}O runs the optimizer every time a certain number of edges have been added. For pose chains this is an inefficient approach as *successive edges* do not add information to the chain that is of value for past nodes. we therefore modified the original code of G^{2}O such that it only runs the optimizer when a *loop-closing edge* is added to the chain. This allows G^{2}O to save a huge amount of unnecessary computations. By setting the number of iterations used by the Gauss-Newton optimizer we can trade-off accuracy and efficiency. During initial experiments we observed that for one iteration of Gauss-Newton G^{2}O provides significantly less accurate results than COP-SLAM while needing significant more computation time than COP-SLAM. For three Gauss-Newton iterations G^{2}O converges to a (local) minima for all our data sets. Three Gauss-Newton iterations are therefore used when a loop-closing edge is fed to G^{2}O.

As our binocular data sets were recorded in an environment with a significantly uneven ground plane, all simulated experiments also focus on 3D environments. Hence, the nodes and edges in pose-chains are elements of SE(3). Our experiments were performed on a single core of an Intel Xeon E5606 CPU (2.13GHz) accompanied by 12 GB of SDRAM. Both COP-SLAM and G^{2}O run equally optimized C/C++ code based on SSE instructions.

A. Simulated data

For our simulation we use 4 different data sets (Fig. 5). For each data set 100 different test trajectories are randomly generated. For the loop and flower data sets the ground truth is based on a template. For the world data sets the ground truth is randomly generated by simulating a vehicle driving over a sphere. Whenever the vehicle comes back to one of its previous locations a loop is detected and the distance between the start and the end of the loop is stored. For one set of experiments we only keep those 5 loops with the closest loop-closing distance (Fig. 5.c) and for another set of experiments we keep 25 loops with the closest loop-closing distance (Fig. 5.d).

On basis of the ground truth 100 trajectories resembling the result of a visual odometer and appearance based loop-detector are simulated. For each of the 100 trajectories and each of the edges in the pose-chain, including the loop-closing edges, a different anisotropic 6×6 covariance matrix is generated. The ratio between the largest eigenvalue of the covariance matrix and its smallest eigenvalue is on average 100. This is similar to what we observe for covariance matrices obtained by linear error propagation inside real visual-SLAM front-ends. According to these covariance matrices random error terms are sampled and added to the ground truth value of the edges (this is all performed using Lie group and manifold methods). For 1 of the 100 experiments the simulated trajectories of each 4 data sets are visualized in the second column of Fig. 5. Although the error per edge is relatively low, the accumulated error (drift) is significant. Again this is similar to what we observe for real visual-SLAM front-ends.

A simulated pose-chain together with the covariance matrices of its edges is then fed to COP-SLAM and to G^{2}O. COP-SLAM can only deal with isotropic uncertainty for the rotational and translational subspaces. Therefore we compute the average variance from the two 3×3 submatrices related to the rotational and translational subspaces respectively. COP-SLAM therefore completely ignores any correlation or anisotropy in the original 6×6 covariance matrix. G^{2}O does take all this statistical information into account giving it an extra theoretical advantage over COP-SLAM for all our experiments.

The results of COP-SLAM and G^{2}O for 1 of the 100 experiments of each data set are visualized in the third and fourth column of Fig. 5 respectively. The average accuracy together with the standard deviation and timing averaged over all 100 experiments for each data set is summarized in Table I. The performance metric is the root mean squared (RMS) error in distance between ground truth absolute poses and their corresponding estimated absolute poses. The error relative to the error of the simulated visual odometer are also provided. This allows for better comparison of results between different data sets. The reported timing values only include time spent on optimization and not time spent on file IO.

From Table I it can clearly be observed that the accuracy of COP-SLAM and G^{2}O is comparable while COP-SLAM is a factor of 50 to 200 times more efficient. For the Loop, Flower and World-25 data sets G^{2}O provides more accurate results than COP-SLAM but for the World-5 data set COP-SLAM is more accurate than G^{2}O. The differences in accuracy relative to the error of the simulated visual odometer range between 0.2% and 2.7% and can be considered marginal. Note that we specifically constructed our simulations such that the experimental conditions are favorable to G^{2}O. The results therefore clearly show that G^{2}O has difficulty utilizing its theoretical advantages on these data sets.

B. Binocular data

To investigate if the same conclusions can be made for real data we recorded three trajectories with a total length of 49 kilometers using a Point Grey Bumblebee2 stereo camera (640x480 color, 6mm focal length, 30 Hertz) mounted on the windshield of a car.

The trajectories go through varied terrain and contain challenging images as can be seen in Fig. 3. The visual odometry

TABLE I

RESULTS ON SIMULATED DATA AVERAGED OVER 100 EXPERIMENTS

	Poses	Loops	Avg. length	RMS VO	RMS C-S	RMS G^{2}O	Time C-S	Time G^{2}O
Loop	10000	1	1 km	143±43 m	61±24 m (42.6%)	**57±23 m** (39.9%)	**4 ms**	297 ms
Flower	8120	8	8 km	308±93 m	48±14 m (15.6%)	**43±13 m** (14.0%)	**6 ms**	1178 ms
World-5	4026	5	400 km	1135±230 m	**557±97 m** (49.1%)	559±98 m (49.3%)	**3 ms**	267 ms
World-25	4026	25	400 km	1131±247 m	254±42 m (22.5%)	**236±38 m** (20.9%)	**30 ms**	1494 ms

TABLE II

RESULTS ON 49 KILOMETERS OF BINOCULAR TRAJECTORIES

	Poses	Loops	Length	RMS VO	RMS C-S	RMS G^{2}O	Time C-S	Time G^{2}O
Pittsburgh A	6927	14	8 km	19 m	**7 m** (36.8%)	**7 m** (36.8%)	**35 ms**	2467 ms
Pittsburgh B	10396	6	14 km	359 m	**68 m** (18.9%)	72 m (20.1%)	**13 ms**	1650 ms
Pittsburgh C	19268	20	27 km	504 m	49 m (9.7%)	**46 m** (9.1%)	**68 ms**	7100 ms

technique used is similar as that used in [5] and provides a robust ML solution by minimizing reprojection errors. Its output are relative pose displacements (i.e. *successive edges*) accompanied by anisotropic 6×6 covariance matrices which are obtained by linear error propagation. Loop-detection was performed by feeding every 10th visual odometry key-frame into RTAB-MAP [16]. Once a loop is detected it is fed to a RANSAC strategy in order to obtain the loop-closing edge. If the number of landmark inliers obtained by the RANSAC strategy is less than 50 the loop detection is rejected. This assures that we only feed reliable and reasonably accurate loop-closing edges to COP-SLAM and to G^{2}O. The pose-chain trajectories that are obtained this way are very representative for those obtained by other state-of-that-art binocular SLAM front-ends in similar environments.

The number of poses and loop detections as well as the results of COP-SLAM and G^{2}O on all three data sets is shown in Table II. The performance metric is the root mean squared error in distance between ground truth absolute position obtained from GPS (with WAAS correction) and their corresponding estimated absolute positions. As the absolute orientation of the first pose of each trajectory is not known with sufficient accuracy, we align each trajectory with the GPS trajectory using the first 50% of its poses. The accuracy is then computed over all poses of a trajectory. We only take the first 50% of the poses when aligning to GPS as to not over fit to the GPS and making the results seem more accurate than they actually are.

The trajectories obtained by GPS, the visual-SLAM front-end, COP-SLAM and G^{2}O are visualized in Fig. 6. The GPS trajectories and those of COP-SLAM are also plotted on top of aerial imagery in Fig. 7.

Again we can observe from Table II that the accuracy of COP-SLAM is comparable to that of G^{2}O while its computation time is again significantly less than that of G^{2}O. On the PittsburghA data set both provide similar accuracy, on the PittsburghB data set COP-SLAM is more accurate and on the PittsburghC data set G^{2}O is more accurate. The differences in accuracy are less than 1% of the error of the visual SLAM front-end and can be considered marginal. In Fig. 6 there are also no significant differences visible between the trajectories of COP-SLAM and G^{2}O.

We can again conclude that from an applied perspective both COP-SLAM and G^{2}O provide the same levels of accuracy. On these data sets COP-SLAM is a factor 100 times faster than COP-SLAM making it applicable to a broader range of applications and computational devices. This makes COP-SLAM either a good and efficient alternative to non-linear iterative optimizers or an excellent closed-form initialization strategy.

VI. CONCLUSIONS

We have presented a novel solution for pose-chain simultaneous localization and mapping called COP-SLAM. At its core it uses an improved version of closed-form trajectory bending and its time complexity is linear in the number of poses. Experiments on a total of 49 kilometers of challenging trajectories estimated by a binocular SLAM front-end show that our solution obtains similar accuracy as optimal non-linear iterative approaches. Our approach however computes its closed-form solution a factor 50 to 200 times faster than these non-linear iterative approaches. This makes COP-SLAM highly applicable to a broad range of robotic applications.

REFERENCES

[1] O. Booij, Z. Zivkovic, and B. Kröse. Sampling in image space for vision based SLAM. In *Robotics: Science and Systems Conference, Inside Data Association Workshop*, 2008.

[2] M. Bosse and R. Zlot. Place Recognition Using Regional Point Descriptors for 3D Mapping. *Springer Tracts in Advanced Robotics*, 62:195–204, 2010.

[3] J. Civera, D. R. Bueno, A. J. Davison, and J. M. M. Montiel. Camera self-calibration for sequential Bayesian structure from motion. In *IEEE International Conference on Robotics and Automation*, pages 403–408. Ieee, May 2009.

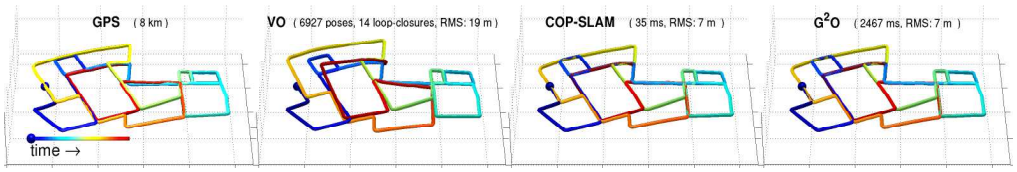
[4] M. Cummins and P. Newman. Appearance-only SLAM at Large Scale with FAB-MAP 2.0. *The International Journal of Robotics Research*, (November), 2010.

[5] G. Dubbelman, L. Dorst, and H. Pijls. Manifold Statistics for Essential Matrices. In *European Conference on Computer Vision*, 2012.

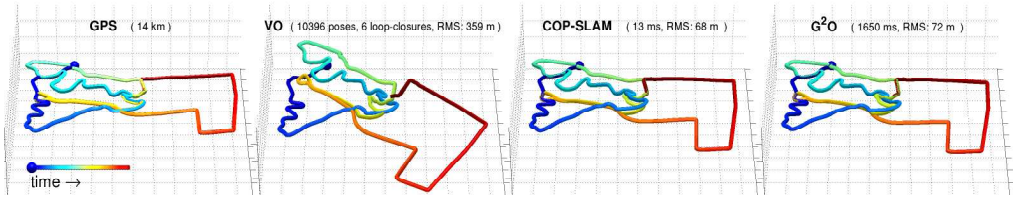
[6] G. Dubbelman, I. Esteban, and K. Schutte. Efficient Trajectory Bending with Applications to Loop Closure. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1–7, Taipei, Taiwan, 2010.

[7] G. Dubbelman, W. van der Mark, and F. C. A. Groen. Accurate and Robust Ego-Motion Estimation using Expectation Maximization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3914–3920, Nice, France, 2008.

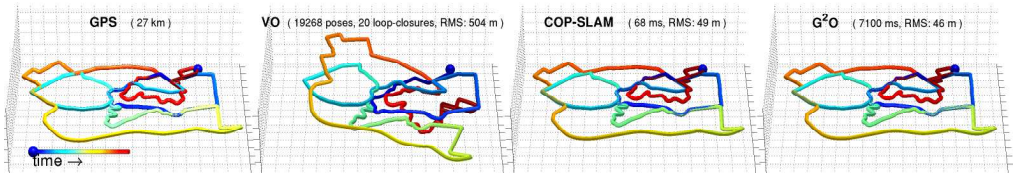
[8] Y. Furukawa and J. Ponce. Accurate, Dense, and Robust Multi-View Stereopsis. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, July 2007.



(a)

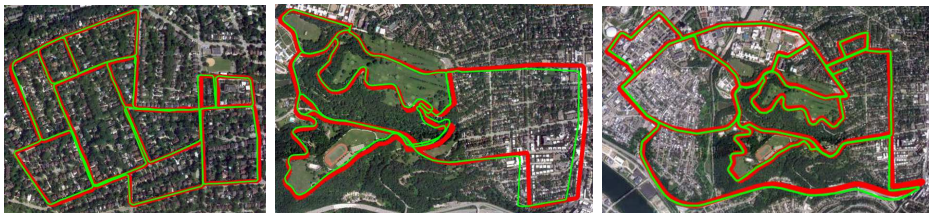


(b)



(c)

Fig. 6. Results plotted in 3D for both methods and all three Pittsburgh data sets. Pittsburgh-A is shown in (a), Pittsburgh-B in (b) and Pittsburgh-C in (c). From left to right: GPS ground truth, visual odometry, COP-SLAM, G^2O. Tile size is 200 meters.



(a) (b) (c)

Fig. 7. Results of COP-SLAM plotted on top of aerial and topographic imagery for all three Pittsburgh data sets. Pittsburgh-A is shown in (a), Pittsburgh-B in (b) and Pittsburgh-C in (c). GPS based ground truth trajectories are plotted in red and trajectories estimated by COP-SLAM in green.

[9] M. Grimes, D. Anguelov, and L. Yann. Hybrid Hessians for Flexible Optimization of Pose Graphs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[10] G. Grisetti, C. Stachniss, and W. Burgard. Non-linear Constraint Network Optimization for Efficient Map Learning. *IEEE Transactions on Intelligent Transportation Systems*, 10(3), 2009.

[11] Y. Jian, D. Balcan, and F. Dellaert. Generalized Subgraph Preconditioners for Large-Scale Bundle Adjustment. In *International Conference on Computer Vision*, number Iccv, 2011.

[12] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *International Journal of Robotics Research*, Dec. 2011.

[13] K. Konolige and M. Agrawal. FrameSLAM: From Bundle Adjustment to Real-Time Visual Mapping. *IEEE Transactions on Robotics*, 24(5):1066–1077, Oct. 2008.

[14] K. Konolige, J. Bowman, J. D. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua. View-based Maps. *International Journal of Robotics Research*, 29(8), 2010.

[15] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A General Framework for Graph Optimization. In *IEEE International Conference on Robotics and Automation*, 2011.

[16] M. Labbé and F. Michaud. Memory management for real-time appearance-based loop closure detection. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1271–1276, 2011.

[17] J. J. Leonard and H. F. Durrant-whyte. Simultaneous Map Building and Localization for an Autonomous Mobile Robot. In *IEEE/RSJ Intelligent Robots and Systems*, pages 1442–1447, 1991.

[18] D. Nistér, O. Naroditsky, and J. Bergen. Visual Odometry. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 652–659, 2004.

[19] E. Olson, J. Leonard, and S. Teller. Fast Iterative Alignment of Pose Graphs with Poor Initial Estimates. In *International Conference on Robotics and Automation*, number May, pages 2262–2269, 2006.

[20] J. M. Selig. *Geometrical Methods in Robotics*. Springer, 1996.

[21] H. Strasdat, J. M. M. Montiel, and A. J. Davison. Real-Time Monocular SLAM: Why Filter? In *IEEE International Conference on Robotics and Automation*, 2010.

[22] N. Sünderhauf and P. Protzel. Towards a Robust Back-End for Pose Graph SLAM. In *International Conference on Robotics and Automation*, 2012.

[23] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[24] R. Wagner, O. Birbach, and U. Frese. Rapid Development of Manifold-Based Graph Optimization Systems for Multi-Sensor Calibration and SLAM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[25] Z. Zhu, T. Oskiper, S. Samarasekera, R. Kumar, and H. S. Sawhney. Ten-fold Improvement in Visual Odometry Using Landmark Matching. In *IEEE International Conference on Computer Vision*, pages 1–8, Oct. 2007.