A *ROS* Implementation of the Mono-SLAM Algorithm

Ludovico Russo, Stefano Rosa, Basilio Bona
Dipartimento di Automatica e Informatica (DAUIN),
Politecnico di Torino, Torino, Italy

Matteo Matteucci
Dipartimento di Elettronica, Informatica e Bioingegneria
(DEIB), Politecnico di Milano, Milano, Italy

***Abstract*— Computer vision approaches are increasingly used in mobile robotic systems, since they allow to obtain a very good representation of the environment by using low-power and cheap sensors. In particular it has been shown that they can compete with standard solutions based on laser range scanners when dealing with the problem of simultaneous localization and mapping (SLAM), where the robot has to explore an unknown environment while building a map of it and localizing in the same map. We present a package for simultaneous localization and mapping in *ROS* (Robot Operating System) using a monocular camera sensor only. Experimental results in real scenarios as well as on standard datasets show that the algorithm is able to track the trajectory of the robot and build a consistent map of small environments, while running in near real-time on a standard PC.**

I. Introduction

In several application scenarios mobile robots are deployed in an unknown environment and they are required to build a model (*map*) of the surroundings, as well as localizing therein.

Simultaneous localization and mapping (SLAM) applications now exist in a variety of domains including indoor, outdoor, aerial and underwater and using different types of sensors such as laser range finders, sonars and cameras [4]. Although, the majority of those approaches still rely on classical laser range finders, the use of vision sensors provides several unique advantages: they are usually inexpensive, low-power, compact and are able to capture higher level information compared to classical distance sensors. Moreover, human-like visual sensing and the potential availability of higher level semantics in an image make them well suited for augmented reality applications.

Visual SLAM approaches are usually divided in two main branches: smoothing approaches based on bundle adjustment, and filtering approaches based on probabilistic filters. The latter are divided in three main classes: *dense*, *sparse* and *semantic* approaches. Dense approaches ([17], [14], [22]) are able to build dense maps of the environment, which make the algorithms more robust but at the same time heavy in terms of computational requirements; indeed, most of these approaches are able to work in real-time only when dedicated hardware (e.g., a GPU or FPGA) is used. Sparse approaches ([15], [7], [12]) address the problem of computational requirements by sparsifying the map; obviously this choice impacts on the robustness of the solution. These algorithms require less computational efforts because they try to allocate in memory only the most significant key points representing the map; for this reason, they are natural candidates for a real time Visual SLAM implementation. Semantic approaches ([10],[20]), extract higher level semantic information from the environment in order to build a more robust and compact map.

Parallel Tracking and Mapping (PTAM) was proposed in [15] as a sparse approach based on a monocular camera targeted to augmented reality applications. The main idea of PTAM is to divide tracking and map updating phases. Camera pose tracking is performed at each time step, by comparing the new frame with the current map using feature matching techniques; *FAST* features [18] are used for matching. Map updating is performed only on a set of keyframes and when the current camera position estimation is precise enough. The algorithm requires an initialization phase in which the same features are viewed from different points of view. The most important limitation of the algorithm is the impossibility to handle occlusions.

A promising solution is the *Mono-SLAM* algorithm, originally proposed by Davison *et al*. in [12]. In this approach, the map and the camera pose are stored as stochastic variables and the system evolution is estimated by an incremental *Extended Kalman Filter* (EKF). *Inverse depth* parametrization can be used for the representation of point features, which permits efficient and accurate representation of uncertainties [16]. By using an approach is known as *active search* paradigm [11], the algorithm is able to speed-up feature matching, since interesting points in the each new frame are looked for only in the most probable regions. In addition, the algorithm does not need an initialization phase and its probabilistic nature makes it more robust to occlusions. The most evident limitation of the algorithm is the fact that, when the map becomes too large, the EKF processing phase becomes too computationally heavy to be computed in real-time [21]. However, some solutions have been proposed to solve that problem too [13].

In this work we present an implementation of the Mono-SLAM algorithm using the ROS [3] framework. The developed ROS node takes as input the images captured from a monocular camera and outputs the trajectory of the camera, as well as a point map representing the environment around the robot. The algorithm is able to run in near real-time on a standard PC with no dedicate hardware for small and medium length trajectories. The rest of the paper is organized as it follows: in Section II we briefly recall the formulation of the Mono-SLAM problem; in Section III we describe our

implementation of the algorithm; in Section IV we show experimental results that validate the effectiveness of the approach; finally in Section V we draw some conclusions and discuss about future extensions of our implementation.

II. Problem Formulation

As in classical SLAM approaches based on laser scanners, the robot pose is described as a stochastic variable with Gaussian distribution, and the map of the environment is sparse. The environment is described by a limited set of features $\{\boldsymbol{f}_i\}$, i.e., measurable geometrical entities (points in the case of Mono-SLAM). Features are described as gaussian variables, as well.

The system state $\hat{\boldsymbol{\mu}}_k$, identified by the robot pose and the map, is represented at any time $t = k\Delta t$, where Δt is the time elapsed since the previous step, as a stochastic variable with Gaussian distribution

$$\hat{\boldsymbol{\mu}}_k \sim \mathrm{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{1}$$

having mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$.

The state vector encapsulates the information on both camera pose and world features

$$\boldsymbol{\mu}_k = \begin{pmatrix} \boldsymbol{x}_k^T & \boldsymbol{f}_1^T & \cdots & \boldsymbol{f}_m^T \end{pmatrix}^T. \tag{2}$$

Information concerning the camera motion at any instant is encoded in the vector $\boldsymbol{x}_k$ built as

$$\boldsymbol{x}_k = \begin{pmatrix} \boldsymbol{r}_k^T & \mathrm{q}_k^T & \boldsymbol{v}_k^T & \boldsymbol{\omega}_k^T \end{pmatrix}^T, \tag{3}$$

where the vector $\boldsymbol{r}$ and the quaternion q represent the pose of the camera reference frame $\mathcal{C}$ while vectors $\boldsymbol{v}$ and $\boldsymbol{\omega}$ are the linear and angular velocities of $\mathcal{C}$ with respect to the world reference frame $\mathcal{W}$.

The function used to predict the evolution of the camera state is given by

$$\boldsymbol{x}_{k+1|k} = \boldsymbol{g}_v(\boldsymbol{x}_k) = \begin{pmatrix} \boldsymbol{r}_k + (\boldsymbol{v}_k + \boldsymbol{V}_k)\Delta t \\ \mathrm{q}_k \times \mathrm{quat}\left((\boldsymbol{\omega}_k + \boldsymbol{\Omega}_k)\Delta t\right) \\ \boldsymbol{v}_k + \boldsymbol{V}_k \\ \boldsymbol{\omega}_k + \boldsymbol{\Omega}_k \end{pmatrix}, \tag{4}$$

where $\mathrm{quat}\left((\boldsymbol{\omega}_k + \boldsymbol{\Omega}_k)\Delta t\right)$ is the quaternion corresponding to the rotation $(\boldsymbol{\omega}_k + \boldsymbol{\Omega}_k)\Delta t$ obtained from the axis-angle representation while $\boldsymbol{V}_k$ and $\boldsymbol{\Omega}_k$ are the noise vectors affecting respectively linear and angular velocities. Features are consider static so they do not need a prediction model.

The measurements function necessary to perform the update step of the EKF filter is given by

$$\boldsymbol{h}(\boldsymbol{\mu}) = \begin{pmatrix} h(\boldsymbol{f}_1, \boldsymbol{x}_k) \\ \vdots \\ h(\boldsymbol{f}_m, \boldsymbol{x}_k) \end{pmatrix}. \tag{5}$$

where the projection function $h(\cdot)$ is a function able to project a 3D features in the image plane using the predicted camera pose.

III. Implementation

The software has been developed using the Robot Operating System (ROS) [3] in C++ under Linux. The OpenCV library [1] was used for image processing and the Eigen3 library [2] was used for matrix operations.

The architecture of the implemented solution is described in Figure 1. Each new frame f_k is acquired from the camera sensor topic by the *capture and preprocess* block, which performs some preprocessing and feeds the frame to the *Mono-SLAM* super-block, which is the main block in charge of processing the frame in order to reconstruct the camera motion and the map. The output of the *Mono-SLAM* main block at each step is the estimated state of the system, as in equation (2).

The *feature matching* block is in charge of matching, in each new frame, the predicted features contained in the set $\mathcal{P}'_{k|k-1}$. In order to improve matching performances, we implemented an active search technique [11]: candidates for new corresponding features are searched only in the most probable areas (which are modeled using ellipsoids), where there is the 99% probability of finding them. This block uses information from the predicted measurements $\hat{\boldsymbol{y}}_{k|k-1}$ and its related covariance $\mathbf{S}_{k|k-1}$ in order compute the measurement vector $\boldsymbol{y}_k$.

When $\boldsymbol{y}_k$ has been computed, the algorithm performs the standard EKF update and prediction steps. The innovation vector $\boldsymbol{e}_k = \boldsymbol{y}_k - \hat{\boldsymbol{y}}_{k|k-1}$ is computed; then the filter performs *update* and *prediction* steps in order to estimate, respectively, the updated state $\boldsymbol{\mu}_{k|k}$ with its covariance $\boldsymbol{\Sigma}_{k|k}$ and the predicted state $\boldsymbol{\mu}_{k+1|k}$ with its covariance $\boldsymbol{\Sigma}_{k+1|k}$. In the proposed implementation, the update step includes the 1-Point RANSAC algorithm [9] for outliers rejection, in order to improve robustness.

The *patches handler* block is in charge of managing the $\mathcal{P}_k$ set. In particular, it is used to find the best features to track at the very beginning (i.e., when the first frame is being acquired) and to delete old features that are no more useful (e.g., when they exit from the frame bounds) and must be replaced. More details are reported in Section III-A.

Blur prediction block is used to predict the motion blur affecting patches when the camera undergoes quick motion. Details are reported in Section III-B.

Note that the z^{-1} block represents a delay equal to the inverse of the camera frame rate, which is used to store predicted information until a new frame has been acquired.

A. Feature matching

The task of features matching consists in finding correspondences for a set of features when a new frame is acquired. Each feature is associated with a descriptor, which will be discussed later in this subsection.

Once a new image is acquired, new meaningful features are found and their descriptors are computed. Then, correspondences are found between the new features and the features from the previous frame. Matching techniques which are fast enough to be used in real-time are usually not robust enough for finding correct correspondences. Luckily, using a

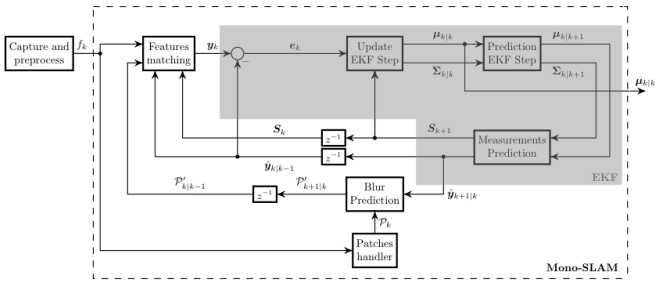


Fig. 1. Architecture of the proposed solution.

probabilistic approach gives an advantage, since the regions where we look for correspondences can be reduced to the ones where the features are more probable to be found. In other words, the search for matches is performed in a region around the prediction of the measurements and the size of the region is related to the covariance matrix of that features; intuitively, the larger the uncertainty on the position of the features, the wider the searching region should be.

In Mono-SLAM, that region is given by projecting the ellipsoid related to the prediction of the feature f_i in the image plane, where s is a number specifying the size of the region, usually $s = 2$ or $s = 3$, i.e., 95% or 99% of probability of finding the feature. Please note that the size of this region is given by $s^2\mathbf{S}_i$, where $\mathbf{S}_i$ is the 2×2 submatrix of $\mathbf{S}_s$ related to the measurements estimation of $\boldsymbol{f}_i$. Hence, the matching is performed on the ellipsoid $\mathcal{S}_i$ having center in p_i and *size* $s^2\mathbf{S}_i$ mathematically described as follows:

$$\mathcal{S}_i = \left\{ p \in \mathcal{I} \middle| \, (p - p_i)^T \, \mathbf{S}_i^{-1} \, (p - p_i) \leqslant s^2 \right\}.$$

If the feature can be matched inside this search region, then it is considered as successfully matched. The image coordinates z_i, denoting the position of the feature in the new frame, are appended to the measurement vector $\boldsymbol{z}_k$ and the feature will contribute towards the correction of the estimates mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$. Otherwise the predicted measurement for $\boldsymbol{f}_i$ will be removed from $\boldsymbol{h}_k$, since there is no corresponding measurement in $\boldsymbol{z}_k$. Subsequently, the corresponding rows of are deleted from the Jacobian $\mathbf{H}_k$ of vector $\boldsymbol{h}_k$. This method, known as *active search*, makes the algorithm to be much more robust and it allows also to use very simple and fast features descriptors to perform matching. For this reason, in this work a simple patch matching techniques was used, as in the original work in [12].

A *patch* is defined as a sub image of given dimensions (usually small) extracted from an image around a specific pixel. In our work the center of each patch is found by an *interest point detector*. Patch matching requires to find the position of the center of the patch inside the original image, i.e., the interest point. The *normalized cross correlation* operator is the default solution to perform patch matching. Given two patches P and P' of the same dimensions, the cross correlation score between the two patches is defined as

$$C_{NCC}(P, P') = \frac{1}{n} \sum_u \sum_v \frac{\left(P(u,v) - \bar{P}\right)\left(P'(u,v) - \bar{P'}\right)}{\sqrt{\sigma_P^2 \sigma_{P'}^2}}, \tag{6}$$

where n is the number of pixels contained in each patch and $\bar{P}$ and σ_P^2 are respectively mean and covariance of the intensity of the pixels of P, defined as

$$\bar{P} = \frac{1}{n} \sum_u \sum_v P(u,v),$$

$$\sigma_P^2 = \frac{1}{n} \sum_u \sum_v \left(P(u,v) - \bar{P}\right)^2 = \frac{1}{n} \sum_u \sum_v P^2(u,v) - \bar{P}^2.$$

Patch matching is performed computing C_{NCC} between the patch to find and each patch having center in the search region $\mathcal{S}_i$, defined above. The center of the patch which maximize C_{NCC} is chosen as best match.

B. Blur correction

When the camera motion is very pronounced, patch matching could fail due to the amount of motion blur affecting the acquired camera frames. To handle motion blur, each patch is pre-blurred using the speed information contained in the predicted state, and the blurred patches are used to perform matching. Unlike alternative solutions that restore the whole image in order to handle blur, this solution uses the Mono-SLAM paradigm in order to reduce computational efforts, by applying blur prediction only to few patches instead of the whole image. More details on the implementation of this approach are reported in a previous work [19].

C. 1-Point RANSAC

1-Point RANSAC algorithm for EKF filters has been originally proposed in [9]. The algorithm is composed by two parts: the first one is in charge of selecting *low-innovation inliers*, while the second one selects *high-innovation inliers*. Low-innovation inliers are selected by executing RANSAC using a single feature (point) to generate hypotheses and select the best consensus set. Unlike classical RANSAC algorithm, the hypotheses are generated using also information given by the EKF filter: new hypotheses are generated by performing EKF update on the selected points only. Then, a consensus set is created by collecting all measured points which lay inside a certain probability ellipsoid (given by a threshold) centered in the predicted measurements obtained from the computed hypothesis.

Low-innovation inliers are elements of the consensus set of the best hypothesis after RANSAC execution. They are assumed to be generated by the true model since they are at a small distance from the most supported hypothesis. The remaining points could be both inliers and outliers, even if they are far from the supported hypothesis. This is due to the fact that the point chosen to generate the best hypothesis could not contain all the information needed to correctly update the state. For instance, it has been explained that distant points are useful for estimating camera rotation, while close points are needed to estimate translation. In the

RANSAC hypotheses generation step, a distant feature would generate a highly accurate 1-point hypothesis for rotation, while translation would remain inaccurately estimated. Other distant points would, in this case, have low innovation and would vote for this hypothesis. But as translation is still inaccurately estimated, nearby points would presumably exhibit high innovation even if they are inliers.

High-innovation inliers are selected after a partial EKF update step involving only the low-innovation inliers selected by RANSAC. After this partial update, most of the correlated error in the EKF prediction is corrected and the covariance is greatly reduced. This high reduction will be exploited for the recovery of high-innovation inliers: as correlations have weakened, consensus for the set will not be necessary to compute and individual compatibility will suffice to discard inliers from outliers. For each point discarded by the RANSAC algorithm, we check if it is individually compatible by verifying whether it lies in a fixed-sized (multiple of its covariance) probability region or not. If the check is passed, the point is added to the high-innovation inliers set. After that all points are checked, a second update involving the high-innovation inliers is performed.

D. Inverse depth coding

In order to tackle the problem of features initialization and features at infinity, in [16], [8] an alternative representation was proposed called *inverse depth*, which represents each feature with six parameters

$$\boldsymbol{\psi} = \begin{pmatrix} x^{\mathcal{C}} & y^{\mathcal{C}} & z^{\mathcal{C}} & \theta & \phi & \rho \end{pmatrix}^T, \tag{7}$$

where $\boldsymbol{r}^{\mathcal{C}} = \begin{pmatrix} x^{\mathcal{C}} & y^{\mathcal{C}} & z^{\mathcal{C}} \end{pmatrix}^T$ is the optical center of the camera the first time the feature is observed, θ and ϕ are respectively *azimuth* and *elevation* of the feature with respect to the image coordinate system and $\rho = 1/d$ is the so-called inverse depth of $\boldsymbol{f}$, where d is the distance of the feature from the optical center the first time it is observed. The 3D position of the features in $\mathcal{W}$ can be computed as

$$\boldsymbol{y} = \boldsymbol{r}^{\mathcal{C}} + \frac{1}{\rho}\boldsymbol{\eta}\left(\theta, \phi\right), \tag{8}$$

$$\boldsymbol{\eta}\left(\theta, \phi\right) = \begin{pmatrix} \sin\theta\cos\phi & -\sin\phi & \cos\theta\cos\phi \end{pmatrix}^T \tag{9}$$

The advantage of using inverse depth encoding is that it allows to compute a normalized vector $\hat{\boldsymbol{u}}$ parallel to $\boldsymbol{y}$ so defined

$$\hat{\boldsymbol{u}} = \frac{1}{d}\boldsymbol{y} = \rho\boldsymbol{r}^{C} + \boldsymbol{\eta}\left(\theta, \phi\right), \tag{10}$$

which is computable also in cases of features at infinity, i.e., with $\rho \to 0$. Note that inverse depth features can not represent point with zero depth, because this implies $\rho \to \infty$. This is not a problem because a features with zero or very small depth implies a real point coincident or very near to the camera optical center, i.e., inside the optics of the camera.

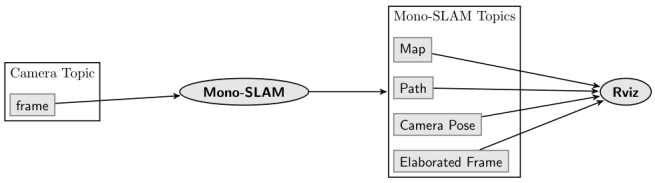


Fig. 2. ROS graph of the developed solution.

(a) (b) (c)

Fig. 3. Examples of the algorithm in action. Positions of the patches are shown in each frame, as well as the ellipsoids that represent their predicted positions in the next frame. (a) shows a standard frame. (b) shows the 1-Point RANSAC algorithm working: blue patches are low-innovation inliers, green patches are high-innovation inliers and red patches are rejected measurements. (c) shows enhances prediction.

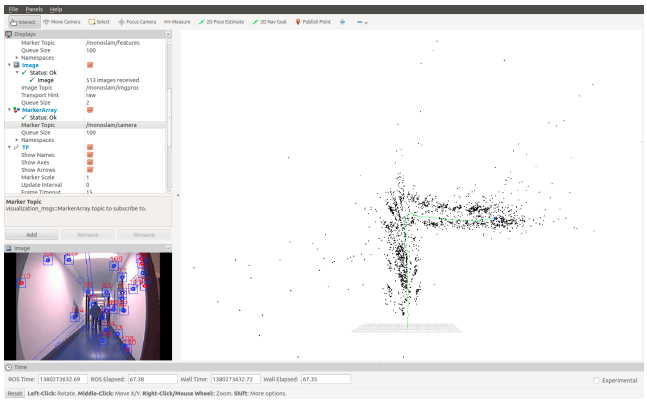
E. ROS implementation details

The algorithm has been fully implemented in ROS as a node called **Mono-SLAM**. The node subscribes to an image topic containing the video stream coming from a video camera (in our work the topic is published by **camera1394** or **gscam** ROS nodes). The full source code for our implementation will soon be available online on the repository of our laboratory. [1]

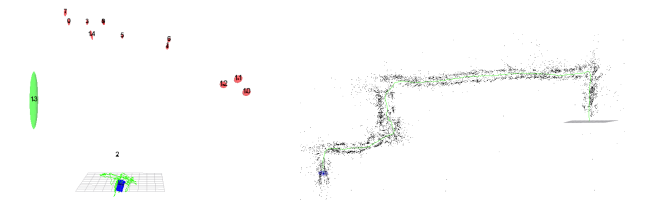
The **Mono-SLAM** node publishes a set of topics which are the camera pose, the camera trajectory, the reconstructed point-map. Other topics which are useful for debugging purposes are the image frames showing the tracked features as well as their state (new, normal, discarded by RANSAC), and the covariances of all features. The topics can be visualized using the **rviz** ROS node. Figure 2 shows the different subscribed and published topics.

Some examples of the node in action are shown in Figure 3 and 4. Figure 3 shows some frames elaborated by the algorithm. Several information are given: correctly matched patches are shown by blue rectangles, features selected as low-innovation inliers are shown as blue points, features selected as high-innovation inliers are shown as green points, features which are discarded by the 1-point RANSAC algorithm are shown as red points and finally the high probability region in which each feature should lie in the next frame is marked by an ellipse. Moreover, Figure 4 shows some examples of the visual output provided by **rviz** ROS node. The camera trajectory and the reconstructed map are shown. Each 3D feature is shown together with its covariance ellipsoid. The green covariance refers to features in inverse depth coding while the red ones refer to features in Euclidian representation. Please note that, due to limits of **rviz**, the covariance of the inverse depth features is represented by projecting the features in the Euclidian space, i.e., always as

[1] `https://github.com/rrg-polito`



(a)



(b) (c)

Fig. 4. Output from the algorithm can be visualized using the **rviz** ROS node. (a) shows a screenshoot of the complete **rviz** environment: camera image with detected features is shown at the bottom-left, while in the main area the resulting 3D map is shown as well as the camera pose and its trajectory (in green). (b) shows the visualization of the ellipsoids representing the covariance on the pose of the features in 3D. Green ellipsoids are associated to inverse depth features while red ellipsoids to Euclidian features. (c) shows a larger map reconstructed by the algorithm.

an ellipsoid, instead of correctly representing that covariance in the inverse depth space, i.e., with a conic shape.

IV. Experimental results

In order to evaluate the proposed algorithm, experimental tests were carried out using a standard monocular camera. Moreover the algorithm has been tested on a standard benchmarking dataset for robotic systems. All the experiments have been carried out on a standard PC equipped with an Intel i7 3.4 GHz CPU and 4 GB of RAM.

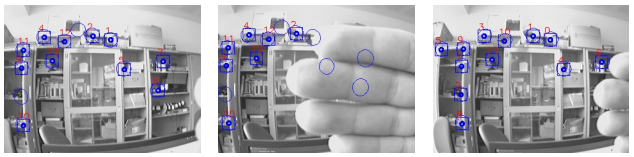
A. Hand held camera

We first tested the algorithm using a simple handheld FireWire camera. We tested robustness to occlusions by introducing an object (hand) in front of the camera while the algorithm was running. Figure 5 shows that the approach is able to reject occluded points and to match them correctly again when the occluding object is removed from the scene.

In another experiment a moving object (person) is present in the scene. In Figure 6 a person is moving in front of the camera. RANSAC is able to rejects moving features, which are not matched. Occluded features are also discarded.

B. Rawseeds dataset

We also tested our algorithm on the datasets freely available from the Rawseeds Project [5]. These are high-quality



(a) (b) (c)

Fig. 5. Testing robustness to occlusions. (a) The algorithm is working normally. (b) Patches occluded by the hand are discarded. (c) The hand is removed from the scene patches are matched again.

(a) (b) (c)

Fig. 6. Testing robustness to moving objects. In (a) the person starts moving. In (b) feature 25 is rejected by RANSAC. In (c) occluded features are removed from the state.

multi-sensor datasets, with associated ground truth, of rovers moving in large environments, both indoor and outdoor. For each sensor, calibration data is provided. For the experiments we used the video stream coming from the front camera of the rover in both indoor and outdoor scenarios.

The robot trajectory estimated by our approach was compared with the ground truth available from the datasets. The ground truth is composed by the trajectory obtained from a multi-camera system and visual tags mounted on the robot [6], which is not available for the whole length of the trajectory, and the estimated trajectory coming from a standard SLAM algorithm based on laser scanner sensors; for the outdoor dataset GPS data is used for the ground truth.

For indoor experiments, we used the *Bicocca_2009-02-26a* dataset [2], in which the robot is moving in an indoor dynamic environment. Some results obtained from this dataset are shown in Figure 7. It is possible to note that the algorithm was able to perform with good accuracy on small and medium scales (10 or plus meters), while on large scales the trajectory is far from the real one. This is due to the fact that the Mono-SLAM algorithm does not explicitly perform *loop closing*. Loop closing only happens when newpatches are correctly matched to previous patches, but this is difficult after large displacements of the camera. Another issue is that the algorithm is not able to correctly measure rotations in some cases. This is due to the lack of significant features in some areas (e.g., when facing a wall) or in presence of too many moving objects in front of the robot.

For outdoor experiments *Bovisa_2008-10-04* dataset [3] has been used. Also this second experiment (see Figure 8) shows that the algorithm works very well on small and medium scales, while on large scales the estimated trajectory drifts from the real one, as in the previous experiment.

[2] http://www.rawseeds.org/rs/datasets/view/6
[3] http://www.rawseeds.org/rs/datasets/view/7

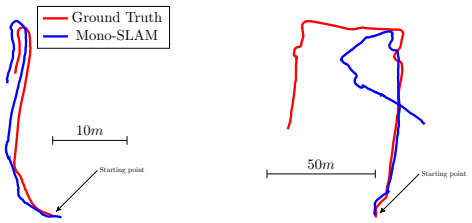


Fig. 7. Results on the *Bicocca_2009-02-26a* dataset. Note that the small errors on estimated angles as well as scale drift both lead to a big difference in the final trajectories.

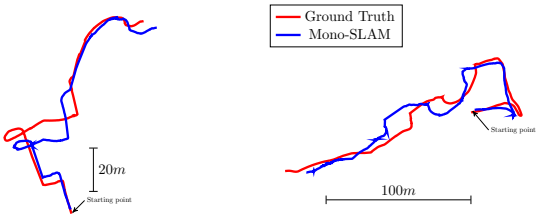


Fig. 8. Results on the *Bovisa_2008-10-04* dataset.

V. Conclusion

We presented a ROS implementation of the Mono-SLAM algorithm. The node is able to run in near real-time and is robust enough on small and medium scales to be used for retrieving the trajectory of the camera as well as reconstructing a 3D point-map of the environment. It should be noted that the performances of the algorithm are heavily influenced by the choice of parameters, in particular noise covariances. Moreover, some robustness issues still remain in the developed algorithm in the case of highly dynamic environments. Some solutions have been implemented in order to improve robustness, and actually the algorithm is able to work in dynamic environments and to correctly manage occlusions. Finally, when the camera trajectory is large, the computational time increases too much for meeting real-time constraints.

Future work will be devoted to improve the robustness of the approach on larger scales by implementing a way to detect and manage loop closings. Moreover, other sensors will be included when available, such as accelerometers, magnetometers, gyroscopes, and wheel odometry. Finally, more complex applications of the Mono-SLAM algorithm are under consideration. These applications concern a multi-robot extension of the algorithm; integration of high-level semantic informations in the algorithm; and an extension of the solution with multi-camera systems.

References

[1] Opencv library. Website. `http://opencv.org`.

[2] Radish: The robotics data set repository. Website. `http://eigen.tuxfamily.org/`.

[3] Ros (robot operating system). Website. `http://www.ros.org`.

[4] Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: a survey. In *Proceedings of the 2008 conference on Artificial Intelligence Research and Development: Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*, pages 363–371, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[5] Andrea Bonarini, Wolfram Burgard, Giulio Fontana, Matteo Matteucci, Domenico Giorgio Sorrenti, and Juan Domingo Tardos. Rawseeds: Robotics advancement through web-publishing of sensorial and elaborated extensive data sets. In *In proceedings of IROS*, volume 6, 2006.

[6] Simone Ceriani, Giulio Fontana, Alessandro Giusti, Daniele Marzorati, Matteo Matteucci, Davide Migliore, Davide Rizzi, Domenico G Sorrenti, and Pierluigi Taddei. Rawseeds ground truth collection systems for indoor self-localization and mapping. *Autonomous Robots*, 27(4):353–371, 2009.

[7] Javier Civera, Andrew Davison, and J Montiel. Dimensionless monocular slam. *Pattern Recognition and Image Analysis*, pages 412–419, 2007.

[8] Javier Civera, Andrew J Davison, and J Montiel. Inverse depth parametrization for monocular slam. *Robotics, IEEE Transactions on*, 24(5):932–945, 2008.

[9] Javier Civera, Oscar G Grasa, Andrew J Davison, and JMM Montiel. 1-point ransac for extended kalman filtering: Application to real-time structure from motion and visual odometry. *Journal of Field Robotics*, 27(5):609–631, 2010.

[10] Javier Civera, Dorian Glvez-Lpez, Luis Riazuelo, Juan D. Tards, and J. M. M. Montiel. Towards semantic slam using a monocular camera. In *IROS*, pages 1277–1284. IEEE, 2011.

[11] Andrew J Davison. Active search for real-time vision. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 66–73. IEEE, 2005.

[12] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.

[13] Ankur Handa, Margarita Chli, Hauke Strasdat, and Andrew J Davison. Scalable active matching. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1546–1553. IEEE, 2010.

[14] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011.

[15] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 83–86. IEEE, 2009.

[16] JMM Montiel, Javier Civera, and Andrew J Davison. Unified inverse depth parametrization for monocular slam. *analysis*, 9:1, 2006.

[17] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE, 2011.

[18] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *In International Conference on Computer Vision*, pages 1508–1515. Springer, 2005.

[19] L. O. Russo, G. Airò Farulla, M. Indaco, Rolfo D. Rosa, S., and B. Bona. Blurring prediction in monocular slam. *International Design and Test Symposium, IEEE Conference*, 2013.

[20] Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. Slam++: Simultaneous localisation and mapping at the level of objects. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, June 2013.

[21] Hauke Strasdat, J. M. M. Montiel, and Andrew J. Davison. Editors choice article: Visual slam: Why filter? *Image Vision Comput.*, 30(2):65–77, February 2012.

[22] Thomas Whelan, Michael Kaess, Maurice Fallon, Hordur Johannsson, John Leonard, and John McDonald. Kintinuous: Spatially extended kinectfusion. 2012.