Blurring prediction in Monocular SLAM

Ludovico Orlando Russo*, Giuseppe Airò Farulla*, Marco Indaco†, Stefano Rosa†, Daniele Rolfo†, Basilio Bona†
Politecnico di Torino
Dipartimento di Automatica e Informatica
* {ludovico.russo, giuseppe.airofarulla}@studenti.polito.it
† {marco.indaco, stefano.rosa, daniele.rolfo, basilio.bona}@polito.it

***Abstract*—The paper presents a method aiming at improving the reliability of Simultaneous Localization And Mapping (SLAM) approaches based on vision systems. Classical SLAM approaches treat camera capturing time as negligible, and the recorded frames as sharp and well-defined, but this hypothesis does not hold true when the camera is moving too fast. In such cases, in fact, frames may be severely degraded by motion blur, making features matching task a difficult operation. The method here presented is based on a novel approach that combines the benefits of a fully probabilistic SLAM algorithm with the basic ideas behind modern motion blur handling algorithms. Whereby the Kalman Filter, the new approach predicts the best possible blur Point Spread Function (PSF) for each feature and performs matching using also this information.**

***Index Terms*—Motion blur, Kernel estimation, Visual tracking, Active vision, Monocular SLAM.**

I. Introduction

Simultaneous Localization And Mapping (SLAM) represents one of the most important and interesting problems in the field of robotic navigation, see [1, chap. 10]. Using SLAM approaches, mobile robots equipped with various sensors are able to build maps of the surrounding environment and, at the same time, to localize themselves in that environment.

Classical SLAM approaches rely on well-known technologies related to robot navigation; for example, laser scanner sensors are very often used to map the environment. However, during the last years, alternative solutions using cheaper and simpler visual sensors have been proposed. This evolution was mainly due to the visual system lower prizes and to the increased computation capabilities of modern computers.

These new approaches to SLAM are intended for low-cost and small systems that may rely only on very limited hardware, such as smartphones. For this reason several solutions have been proposed to reduce and simplify data processing, as in [2].

In computer vision, the most similar approach to SLAM is the Structure from Motion (SfM) problem, used in [3]. It allows reconstructing both camera movements and the scene by off-line processing the video stream; moreover, it can use a large number of robust features detected in the scene at every frame and match them frame by frame. The most important limitation in SfM algorithms is the fact that they do not assume simple and small camera displacements over two sequential frames, while they consider camera point of view independently in each frame. In other word, the task of features matching is always done using very robust features that require sophisticated hardware resources with high computational capabilities to be computed. For this reason, this approach is not suitable for embedded and low-cost devices.

Davidson at al. in [4] have developed, in the last years, a very efficient SLAM algorithm based on a single camera and the Extended Kalman Filter (EKF) algorithm. Their work is based on probabilistic robotics navigation approaches, and they take care of efficiency especially when building maps and matching features, operations that can be done in real time. The most significant innovations introduced in Davidson's approach are (i) the construction of a sparse but meaningful map of the environment to reduce memory, and (ii) the processing of resources in order to use knowledge of the camera motion predicted by the EKF. In such a way, it is possible to reduce features matching regions. Especially with regards to this second point, the *active matching* approach [5, 6] was proposed to speed up the features matching phase in the algorithm.

While Davidson's and related work deals in particular with the algorithmic part of the problem, in terms of EKF development, features matching, etc., they all make an implicit assumption on the vision device. In all these approaches, in fact, the frame capturing time is considered negligible with respect to the camera motion. This assumption holds well only when the camera itself is moving slowly. On the other hand, it can cause major issues in features matching when the camera is moving with a sustained speed, because recorded frames may be blurred. Classical SLAM approaches cannot be properly applied on blurred frames, since features upon them will not be recognizable and trackable. This represents a problem because mobile robots may totally lose the orientation, and so the possibility of building a map of their surrounding environment, even just failing the feature matching task in two consecutive frames. Knowing that, it is crucial to avoid as much as possible the disastrous effects of motion blur on SLAM algorithms.

Aim of this paper is to apply results of computer vision coming from deblurring theory to extend the Mono-SLAM algorithm in case of high speed motion of the camera.

The paper is organized as follows: Section II presents an overview about the Mono-SLAM algorithm and problems related to blur, Section III discusses related works, Section IV introduces the problem from a theoretical point of view, Section V presents the proposed solution, Section VI discusses the obtained experimental results and Section VII presents the conclusions and possible future works.

II. Mono-SLAM and Deblurring: An Overview

A. Mono-SLAM

Mono-SLAM is an algorithm proposed by *Davidson at al*. in [4] which is one of the most relevant solution to the SLAM problem using monocular vision systems.

As in classical SLAM approaches based on laser scanner sensors, the robot pose is described as a stochastic variable with Gaussian distribution, while the map of the environment is sparse. The environment is described by means of a limited set of "features" $\{\boldsymbol{f}_i\}$, i.e., measurable geometrical entities. Examples are points (as in classical Mono-SLAM) or lines (as in [7]). Features are described as stochastic variables, as well.

The system state $\hat{\boldsymbol{\mu}}_k$, identified by the robot pose and the map, is represented at any instant $t = k\Delta t$, where Δt is the camera frame-

rate, 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 of both camera and world state

$$\boldsymbol{\mu}_k = \begin{pmatrix} \boldsymbol{x}_k^T & \boldsymbol{f}_0^T & \cdots & \boldsymbol{f}_m^T \end{pmatrix}^T. \tag{2}$$

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 speed 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 $(\boldsymbol{\omega}_k + \boldsymbol{\Omega}_k)\,\Delta t$ in axis-angle representation while $\boldsymbol{V}_k$ and $\boldsymbol{\Omega}_k$ are the noise vectors affecting respectively linear and angular speed. Features are consider static so they do not need a prediction model. For more information about the algorithm, please refer to [4, 8, 9].

B. Blur handling

The problem of restoring a blurred image has long been a challenging problem in digital imaging. It has been studied in depth from several points of view; a simple and quick research over the literature reveals that many studies have been done on this argument and hundreds of papers have been written during the last four decades (for instance [10], [11], [12]). Several studies have focused on the task of recovering a latent image starting from an input motion blurred one. Very often, authors have modelled the task as a deconvolution process. This simplification holds on when the blur is considered spatially invariant (or shift-invariant), meaning that every point in the original image spreads out the same way in forming the blurry image. A commonly used notation is the following one ([13]):

$$f = g * p + n \tag{5}$$

where $*$ is the discrete two dimensional convolution operator, g is the original image to recover (i.e., the one that would have been observed if no blur or noise occurred), f is the observed blurry and noisy image, p is the blur kernel (or Point Spread Function, PSF), and n is the noise. It is common to model n as a Poisson or white Gaussian noise, uncorrelated with the true image g, although this consideration does not always hold true [14].

The problem of recovering the latent image g from (5) is well known to be ill-conditioned, as even small variances in the modelled kernel p leads to errors in the observed output f. The introduction of an error Δp will cause the (5) to be reformulated as

$$f = g * p + g * \Delta p + n, \tag{6}$$

thanks to the linearity of the convolution operator. In some cases, the contribution of $g * \Delta p$ is not negligible, mainly if high quality outcomes are expected from the deblurring process.

However, errors in blur kernel estimation may be drastically reduced if reliable information is deducible from the context. For example, p can be calculated with high precision if it is well known how, and how fast, the camera is moving. Moreover, if the camera is equipped with an high quality sensor, the noise components directly related with the camera (e.g., shot and random noise) are drastically reduced, while the environmental components of n can be managed by an appropriate pre-processing phase.

III. Related works

This paper relates to the literature on both Mono-SLAM and blur handling. The amount of works produced in the last four decades on both these topics is too wide to be summarized here. So, this small analysis about the reference state-of-the-art only addresses works directly related to blur handling approaches used to somehow improve and ease SLAM tasks using visual systems.

The approach discussed in [15] refers to handling motion blur in SLAM. It works by deblurring every video frame using estimations about camera movements derived from 3-D point structures reconstructed by SLAM. In such a way, authors state that SLAM and deblurring are carried out simultaneously, and improve each other's results. Even if authors use the same model of blur and a similar way of estimating its parameters as it is done in this paper, their approach relies on very complex and time-consuming operations, required to deblur whole images.

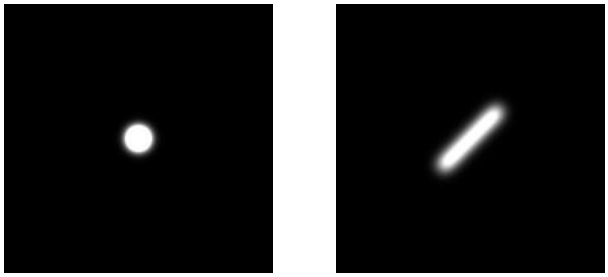
The approach proposed by *Lee et al*. presents some limitations that the proposed method overcomes: it may worsen the results of the localization task when wide camera movements are considered (or even when there is no blur at all); moreover, to fulfill real-time constraints, it must use two different threads for mapping and localization and GPU-based acceleration for the operations of deconvolution.

In [16] the problem of tracking features in presence of motion blur is addressed. Authors model motion blur as a Gaussian convolution process. After estimating blur parameters, the template image is convolved and so blurred to estimate the real shape of the features to track. Performances and reliability of this approach may be improved by using derivatives and considerations about camera shutter timing, as in [17]. While these approaches present some similarities with the algorithm here presented, they *simply* solve the task of features tracking. SLAM is a more complicated problem, as it uses features tracking to build maps of unknown environments and to contemporaneously identify camera poses.

IV. The proposal: blurring correction applied to Mono-SLAM

To track features within a group of images (e.g., frames of a video stream), Mono-SLAM algorithms compare pixels with one (or more) patch, and matches, if found, are highlighted. However, in some cases, camera movements may be so pronounced, and so it would be hard for any classifier to recognize a feature even if it is known, with a reasonable uncertainty, where the feature itself is located. This problem is caused by motion blurring. To give an example, imagine that the feature to be recognized is represented by a point, e.g., a star in the sky as in figure (1a), and the input of the SLAM algorithm is captured by a camera translating very fast. In this case, the illusory movement of the considered scene will be too large even if the camera is recording with a high temporal resolution (i.e., the shutter time is small). So, it is probable for the expected point to be actually recorded as it were a straight line, as in figure (1b). In such a case, it will be impossible to recognize it.

Classical and related works in the field of robotic SLAM always consider negligible the shutter time and frames to be tracked as perfectly on focus. The main contribution of the novel approach here proposed is to overcome this limitation, thus allowing an easy identification of blurred features.



(a) Original image (b) Blurred image

Figure 1: Sharp (a) and blurred (b) version of a star in the sky.

One possible approach to solve the motion blurring problem related to Mono-SLAM could be to pre-process each recorded frame by a deblurring filter, to recover the real latent image behind it. After that, it could be possible to identify and match features even using regular and sharp patches. However, this approach will be overly time consuming, while Mono-SLAM algorithm is required to fulfil real-time constraints. In addition, obtained results may be unacceptable, since real blur effects due to camera movements (or shake) may hardly be perfectly modelled. Similar considerations hold on even if considering narrowing the deblur problem to the regions in the frame in which features are expected to be found.

In this paper, problems related to blur effect are addressed from a novel point of view. In fact, they have been studied in order to ease the task of Mono-SLAM, and at the same time improving its reliability. Main purpose of Mono-SLAM is determining the position and orientation of the camera by analysing images captured by it. Taking this into account, the approach presented in this paper proposes to blur the patches used to recognize features, in order to line up observed features and their expected appearance. This novel approach presents various advantages:

- considered patches are very small and in such tight spaces the hypothesis of spatial-invariance can be applied without loss of generality;
- the operation of convolution modelled by (5) can be performed very quickly;
- information about rotational and translational speed given by the EKF are used to deduce the PSF, and so to calculate the expected feature shape through a simple convolution – in the same way as it is expressed in (5).

A. Blur kernel prediction

To predict the kernel used to blur (when needed) patches, the following assumption have been made: (i) the camera capturing time T is constant and known, (ii) the blur kernel is linear. This second hypothesis is perfectly suitable for small patches (e.g., 21×21 pixels as in the experiments presented in section VI).

The matrix representing the PSF is deduced for each frame from information about speed given by (4). For the subsequent blurring task the kernel is identified by a sparse matrix, with non-zero values identifying a straight line representing the direction of camera movements. The length and direction of this line is calculated by computing the $3D$ displacement of the interest point $\boldsymbol{p}$ in the capturing time interval T using the estimated linear and angular speed as follows

$$\boldsymbol{p}^* = \begin{pmatrix} \mathbf{R}(\boldsymbol{\omega} T) & \boldsymbol{v} T \\ 0 \cdots 0 & 1 \end{pmatrix} \boldsymbol{p} \qquad \rightarrow \qquad \Delta \boldsymbol{p} = \boldsymbol{p}^* - \boldsymbol{p}, \tag{7}$$

and projecting it on the image plane using the projection function $\underline{h}\,()$ as follows

$$\Delta \underline{p} = \underline{h}\,(\boldsymbol{p}^*) - \underline{h}\,(\boldsymbol{p})\,, \tag{8}$$

see also fig. (2). $\mathbf{R}(\boldsymbol{\omega} T)$ is the rotation matrix associated to the angle $\boldsymbol{\omega} T$.

The PSF is the smallest possible matrix containing the $\Delta \underline{p}$ and presenting non-zero entries along $\Delta \underline{p}$ direction. Non-zero values have to be normalized in order to make their sum equal to one; this will ensure that the convolution process will not affect the luminosity of the patch (otherwise the result would be brighter or darker).

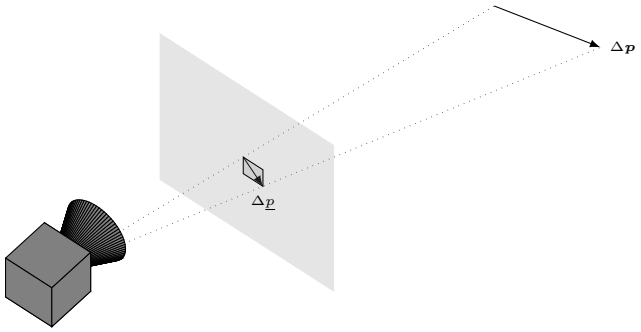


Figure 2: Kernel computation: the displacement of the considered point $\Delta \boldsymbol{p}$ during the shutter time T is projected on the image plane. The blurring kernel (the dark gray rectangular) is the smallest matrix containing entirely the projection $\Delta \underline{p}$.

B. Blurring prediction algorithms

Algorithm (1) presents the pseudo-code of the operations that are used in the proposed solution to predict the blurred patch. It requires as input information contained in the predicted state $\boldsymbol{\mu}_{k+1|k}$. It performs the following operations: $\Delta \underline{p}$ is computed as in (8) and its length is compared with a threshold to verify if the blurring is not negligible (line 1), if blurring is not negligible (line 2), the patch is blurred (lines 3, 4) and the result of this operation is stored into a new patch p' (line 5); otherwise, p' will be equal to p.

Algorithm 1 Blur prediction

Input: patch p, predicted state $\boldsymbol{\mu}_{k+1|k}$
Output: predicted patch p'
1: $\Delta \underline{p} \leftarrow compute\_displacement\left(p, \boldsymbol{\mu}_{k+1|k}\right)$
2: **if** $\|\Delta \underline{p}\| > MIN\_SIZE$ **then**
3: $\quad PSF \leftarrow compute\_kernel\left(\Delta \underline{p}\right)$
4: $\quad p' \leftarrow blur\_patch\,(p, PSF)$
5: **end if**

C. Architecture

The developed architecture is described in figure 3. The camera used as input device is sending captured frames to the *capture and preprocess* block, that allocates the captured frame f_k and sends it to the *Mono-SLAM* main block, that is in charge of processing the

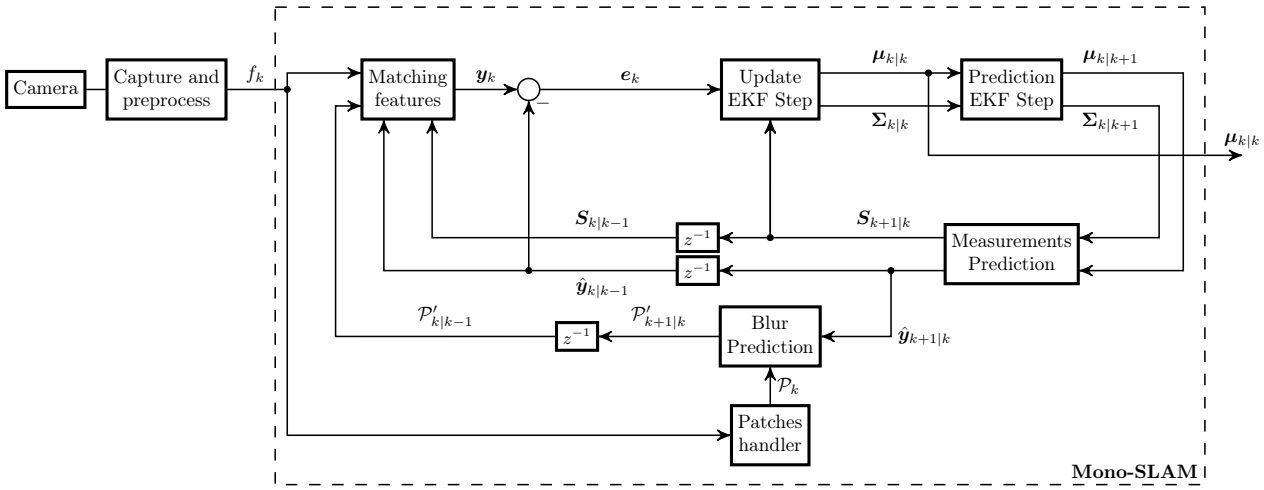


Figure 3: Architecture of the proposed solution.

frame to reconstruct the camera motion. The output of the *Mono-SLAM* main block is the estimation of the state of the system, as in equation (2).

The *matching feature* block is in charge of matching the predicted features contained in the set $\mathcal{P}'_{k|k-1}$ in the new frame. For performance reasons, matches for all features are searched in the areas (modelled as ellipsoid) in which there is the 99% probability of finding them. This block uses information contained in the predicted measurements $\hat{\boldsymbol{y}}_{k|k-1}$ and its related covariance $\mathbf{S}_{k|k-1}$. Its output is the measures vector $\boldsymbol{y}_k$.

When $\boldsymbol{y}_k$ has been computed, the algorithm performs the standard EKF filter. Firstly, the innovation vector $\boldsymbol{e}_k = \boldsymbol{y}_k - \hat{\boldsymbol{y}}_{k|k-1}$ is computed; secondly, the *update* and *prediction* steps are performed to build, respectively, the updated state $\boldsymbol{\mu}_{k|k}$ with related covariance $\boldsymbol{\Sigma}_{k|k}$ and the predicted state $\boldsymbol{\mu}_{k+1|k}$ with related covariance $\boldsymbol{\Sigma}_{k+1|k}$.

The predicted state is used to build, for each patch in the set $\mathcal{P}_k$, the set of blurred patches $\mathcal{P}'_{k+1|k}$ expected in the next frame by iteratively run the algorithm (1) for all elements in the set $\mathcal{P}_k$.

The *patches handler* block is encharged 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 when old patches have been deleted because they are no more useful (e.g., when they go outside the framing) and must be replaced.

Finally, note that z^{-1} represents a delay equal to the inverse of the camera frame rate, useful to store predicted information until a new frame has been acquired.

V. Implementation

The Mono-SLAM algorithm and its improvements here presented has been implemented using the Robotic Operating System (ROS) framework [18], version Groovy, in C++ under Linux Ubuntu OS. Mathematical computations required by EKF are performed using the Eigen3 open source library [19], while image processing is performed using the OpenCV library, v2.4 [20]. The camera used in the experiment is a firewire camera with a temporal resolution of $30fps$ and a spatial one of 340×480 pixels in gray scale. The model used to compute distortion and projection is the same as in the Camera Calibration Toolbox of Matlab [21], which is used to calibrate the camera.

The algorithm is implemented as a ROS node which subscribes to a topic publishing camera frames. By means of a callback, a function implementing the EKF algorithm is triggered when a new frame is captured. When the very first frame is captured, the algorithm proposed in [22] and implemented in OpenCV is used to detect a certain number (defined via an input parameter) of good features to track. In addition, for comparison purpose, a flag is used to enable, or disable, the functions managing blur. In addition, also the variables necessary for the EKF, like initial and noise covariance, and camera calibration parameters, are fixed at launch time through a configuration file.

The ROS node also publishes messages containing the current pose of the camera and the position of each feature in the world reference frame, that can be later visualized through the built-in program RVIZ. Moreover, the node publishes a message containing the current frame in which the matched features are highlighted and, for each of them, the corresponding ellipsoid in which the feature is supposed to be located in the next frame. Features are colored by the RANSAC algorithm; in particular using red if they were rejected by low inlier test, blue if accepted as members of $\mathcal{I}_L$ but rejected after the high-inlier coherency check and green otherwise. All the comparisons between the base algorithm and the proposed solution are done using the same videos.

VI. Experimental Results

To show the effectiveness of the proposed solution, several tests have been carried out over different videos.

These tests show that real-time performance can be achieved tracking up to 30 features per frame. The test machine is a desktop PC equipped with an Intel i7-2600K CPU @3.4GHz and 8 GB of RAM. Moreover, a session of tests has been devoted to test the effectiveness of the proposed approach in a simulation of a realistic field of usage, with the camera that is moving approximately at constant speed and with 30 features to track. Several videos have been recorded, with the camera moving at different speeds. In addition, the proposed algorithm has been executed with the same video input more than once, to test its performances with respect to variations in the input value for camera occlusion time T.

The test program executes the proposed solution but also computes matching with the original (i.e., not blurred) patch as the standard Mono-SLAM algorithm, and it stores the two cross-correlation values, c and c^* respectively, together with the punctual relative patch speed in the image, computed by the EKF algorithm. Notice that if $c > c^*$ the proposed method achieves better results then the original one, while the other way around if the condition $c < c^*$ holds true.

To highlight the goodness of the proposed approach, Figure (4) shows a plot of the index $r = c/c^*$ with respect to the relative patch speed (measured in *pixel/frame*, that indicates the displacement of the patch between two subsequent frames) and an occlusion time set at $T = 3.5ms$. As expected, when features are slowly moving this ratio is almost always equal to one, as the effects of blur are less noticeable. On the other hand, the faster is moving the feature, the better the proposed solution works. For this reason, with the exception of few outliers, the ratio is always bigger than one.

Indexes describing the overall performances of the proposed solution and the original one are here presented. N indicates the percentage of accepted matches with blur handling, while N^* represents the same value but when the blur is not considered. $\xi_>$, $\xi_=$ and $\xi_<$ indicate the percentage of cases in which the value c is, respectively, greater, equal or less then c^*, that is when the proposed method performs better, at the same way or worse then the original one. These values are presented in Table I with respect to different hypothesized shutter time T. The indexes are also reported in the overall case and differentiated according to the relative speed of the patch, i.e., low speed (less than 6*pixel/frame*), high speed (more than 12*pixel/frame*) and medium speed.

Results show that the proposal performs better in particular for shutter time T between $4.5ms$ and $6ms$. Notice that performances of the original algorithm are actually similar to the ones achieved by the new proposal when features are moving slowly, while they decrease when features are moving faster. Moreover, note that the index N is almost invariant to the relative speed, indicating that the proposed approach is robust to fast motion. On the other hand, this is not true for the classical approach, since values of N^* considerably decreases when speed increases.

The proposed algorithm almost never achieves performances that are worse than the ones achieved by the classical approach; when it happens, it is just because the EKF is intended to work on frames recorded by a camera moving at a costant speed, and so it may return uncorrect previsions when the camera is momentaneously accelerating. Instead, when the blur is properly handled the percentages of features correctly matched increases considerably, and the measured cross-correlations indicated that uncertainties on features matching are significantly reduced.

Note that the proposed solution can not handle high accelerating motions, since the EKF filters can not accurately predict the new velocities because the consider model is a constant speed model. In this case, the motion blur is badly estimated. However, this cases can not be handled even by the original approach.

The basic algorithm usually performs better in the very first frames, because the EKF filter is unable of accurately predicting features location; for this reason, the blur kernel cannot be precisely calculated, and so blurring is performed incorrectly. However, after this first initial phase, the predictions given by EKF filter converge to the exact features location. After this phase, the blur kernel can be more precisely estimated, and can be so used to strengthen the features matching step; on the other hand, the basic SLAM solution is not robust to quick camera movements or rotations.

On the above mentioned test machine, a typical breakdown of the processing time required at each frame at 30Hz is described in Table II.

This indicates that 30Hz performance is easily achieved, as 33 ms are available for processing each frame. An optimized version of the code should achieve 60Hz performance.

VII. Conclusion

In this paper a novel deblurring approach to improve the robustness of Mono-SLAM algorithms has been proposed. This approach uses information given by the EKF to estimate the blur kernel and cleverly blur expected patches to ease the features matching task. The theoretical justifications of this approach rely on the fact that motion blur should not be ignored even when small patches are considered (as it is currently done by the state of the art algorithms), while it can be treated as spatially-invariant. The proposed algorithm has shown to overcome performances obtained by classical Mono-SLAM algorithms, that are limited by treating as negligible camera occlusion time. Moreover, it uses the simplest possible linear model for blur related operations, and so it can fulfill real-time constraints.

Possible future works may be related to define an uncertainty threshold above which it is useless to consider blur, while classical SLAM algorithms perform better. More complex model to handle also spatially-variant motion blur or big amount of noise may be studied, even if so far tests have not highlighted this necessity. Finally, test sessions with humanoid robots have already been planned to further demonstrate the goodness of the proposed approach.

References

[1] Sebastian Thrun, Wolfram Burgard, Dieter Fox, et al. *Probabilistic robotics*. Vol. 1. MIT press Cambridge, 2005.

[2] S.S. da Costa Botelho et al. "Visual odometry and mapping for Underwater Autonomous Vehicles". In: *Robotics Symposium (LARS), 2009 6th Latin American*. 2009, pp. 1–6. DOI: 10.1109/LARS.2009.5418320.

[3] P. Woock et al. "Odometry-Based Structure from Motion". In: *Intelligent Vehicles Symposium, 2007 IEEE*. 2007, pp. 1112–1117. DOI: 10.1109/IVS.2007.4290266.

[4] Andrew J Davison et al. "MonoSLAM: Real-time single camera SLAM". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 29.6 (2007), pp. 1052–1067.

[5] Margarita Chli and Andrew Davison. "Active matching". In: *Computer Vision–ECCV 2008* (2008), pp. 72–85.

[6] Margarita Chli and Andrew J Davison. "Active matching for visual tracking". In: *Robotics and Autonomous Systems* 57.12 (2009), pp. 1173–1187.

[7] Paul Smith, Ian Reid, and Andrew Davison. "Real-time monocular SLAM with straight lines". In: *British Machine Vision Conference*. Vol. 1. 2006, pp. 17–26.

[8] Javier Civera, Andrew Davison, and J Montiel. "Dimensionless monocular SLAM". In: *Pattern Recognition and Image Analysis* (2007), pp. 412–419.

[9] Javier Civera, Andrew J Davison, and J Montiel. "Inverse depth parametrization for monocular SLAM". In: *Robotics, IEEE Transactions on* 24.5 (2008), pp. 932–945.

[10] O. Whyte et al. "Non-uniform Deblurring for Shaken Images". In: *International Journal of Computer Vision* 98.2 (2012), pp. 168–186.

[11] Wei Wang and Michael K Ng. "On algorithms for automatic deblurring from a single image". In: *Journal of Computational Mathematics* 30.1 (2012), pp. 80–100.

[12] Xiaogang Chen et al. "An effective document image deblurring algorithm". In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE. 2011, pp. 369–376.

Table I: Results obtained during the test sessions with respect to different occlusion time T (expressed in $[ms]$). The results are reported globally and considering the speed of the patches. Note that the new proposal performs at the same way regardless the relative speed of the patch, while the original algorithm get worse when speed increases.

		General				**Low speed**				**Medium speed**				**High speed**			
T	$[ms]$	4.5	6	7.5	9	4.5	6	7.5	9	4.5	6	7.5	9	4.5	6	7.5	9
$\xi_>$	[%]	74.3	74.3	73.3	70.0	18.3	25.6	30.3	32.8	96.0	95.1	91.9	85.5	99.6	97.8	97.5	97.4
$\xi_=$	[%]	19.8	17.5	16.5	16.0	66.6	55.8	49.4	43.6	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$\xi_<$	[%]	5.9	8.2	10.2	14.0	15.1	18.6	20.3	23.6	3.8	4.9	8.1	14.5	0.4	2.2	2.5	2.6
N	[%]	99.4	99.3	98.3	98.1	99.3	99.0	98.9	98.5	99.6	99.5	97.7	98.7	99.2	99.4	98.1	97.1
N*	[%]	88.2	86.9	86.1	86.1	99.0	98.6	99.2	98.7	93.3	91.2	87.8	87.7	74.6	73.8	71.2	70.3

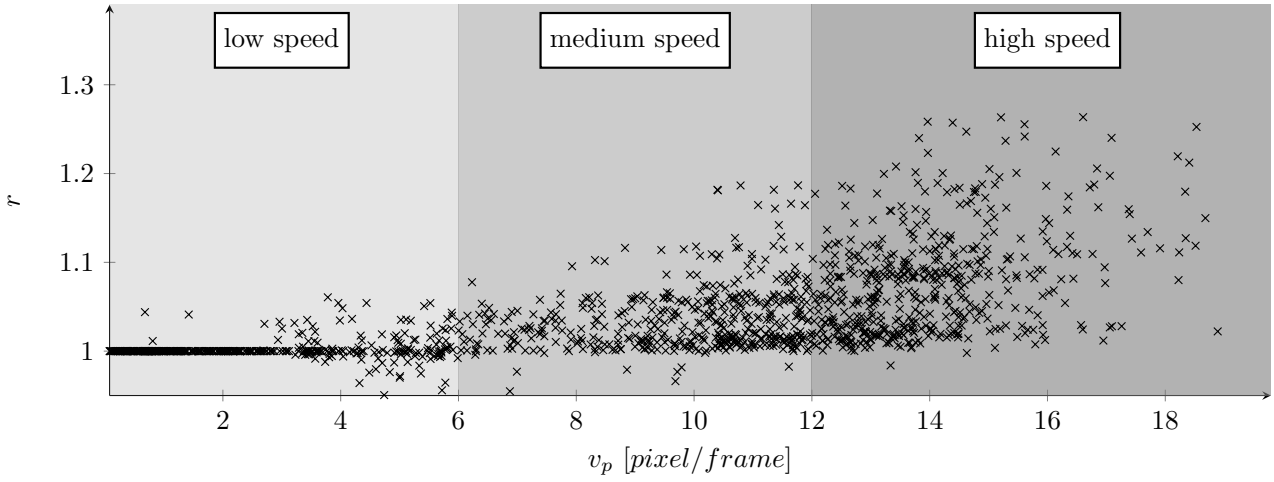


Figure 4: Ratio $r = c/c^*$ of the two best matching correlation score of the same patch with and without blur prediction over different camera speed assuming an occlusion time $T = 6ms$. Note how the proposed solution performs better when patch relative speed increases.

Table II: Average results of timing profiling operations performed over the implemented code.

Image loading and administration	2 ms
Image correlation searches	3 ms
Kalman Filter update	5 ms
Feature initialization search	4 ms
Graphical rendering	5 ms
Blur kernel prediction	1 ms
Convolution operations	1 ms
Total	21ms

[13] Jian-Feng Cai et al. "Framelet-Based Blind Motion Deblurring From a Single Image". In: *Image Processing, IEEE Transactions on* 21.2 (2012), pp. 562–572.

[14] Patrizio Campisi and Karen Egiazarian. *Blind image deconvolution: theory and applications*. CRC press, 2007.

[15] Hee Seok Lee, Junghyun Kwon, and Kyoung-Mu Lee. "Simultaneous localization, mapping and deblurring". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. 2011, pp. 1203–1210. DOI: 10.1109/ICCV.2011.6126370.

[16] Hailin Jin, P. Favaro, and R. Cipolla. "Visual tracking in the presence of motion blur". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 2. 2005, 18–25 vol. 2. DOI: 10.1109/CVPR.2005.372.

[17] Youngmin Park, V. Lepetit, and Woontack Woo. "ESM-Blur: Handling & rendering blur in 3D tracking and augmentation". In: *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*. 2009, pp. 163–166. DOI: 10.1109/ISMAR.2009.5336480.

[18] *ROS*. URL: http://www.ros.org/wiki/.

[19] *Eigen3 library*. URL: http://eigen.tuxfamily.org.

[20] *OpenCV library*. URL: http://opencv.willowgarage.com/wiki/.

[21] *Camera Calibration Toolbox for Matlab*. URL: http://www.vision.caltech.edu/bouguetj/calib_doc/.

[22] J. Shi and C. Tomasi. "Good features to track". In: *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*. 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.