Visual Odometry Parameters Optimization for Autonomous Underwater Vehicles

Pep Lluís Negre Carrasco, Gabriel Oliver-Codina
Systems, Robotics and Vision Group,
University of the Balearic Islands
Cra Valldemossa km 7.5 (07122),
Palma de Mallorca, Balearic Islands
pl.negre , goliver @uib.es

Abstract — *Visual odometry algorithms are used in a wide range of applications to provide reliable estimates of the movement of the camera frame. Combined with inertial measurement units and other low-cost sensors, they can be used as input of high-level algorithms like Kalman filters or SLAM to estimate the global vehicle pose. However, visual odometers are often complex algorithm pipelines with large parameter sets involved in every process stage. In this paper we focus on optimizing the parameters of a visual odometer that has proved to work in underwater environments. We present the results of an exhaustive optimization process to reduce the errors in rotation and translation comparing vision-based pose estimates with ground truth in real underwater environments.*

Keywords—Visual Odometry, Underwater Vehicles, Stereo Images, Motion Estimation

I. INTRODUCTION

Although there are many visual odometers in the literature, we focus our work on visual odometry systems that use stereo cameras in order to obtain precise 6 DOF camera poses. The common algorithm pipeline [1] for stereo visual odometers is based in the following steps: first, keypoints (landmarks) are identified in each camera frame and feature descriptors for these points are extracted. Then, the depth for every landmark is estimated using stereo, structure from motion or a separate depth camera. Subsequently, features are matched across time frames and the rigid-body transformation that best aligns the features between frames is estimated. The result of this process is an estimation of camera motion between frames and therefore it is necessary to integrate this data over time to obtain the vehicle's absolute position and orientation.

The previous work [2], developed by members of our group, compares two publicly available visual odometers (*libviso2* [3] and *fovis* [4]) for use in autonomous underwater vehicles (AUVs). In [2] it is also described the main features of both odometers and its performance is compared from the results carried out using two AUVs in real sea environments.

However, each visual odometer has different algorithm stages with its own parameters that must be configured for every application. The present work compares the main parameters of *libviso2* and *fovis* in order to find equivalences and highlight the most important ones. Then, the effect of each parameter on the final odometry performance is studied. Finally, the work investigates possible correlations between the parameters in order to find more effective ways to configure these visual odometers.

II. VISUAL ODOMETRY PARAMETERS

A visual odometry algorithm is a complex program with a pipeline containing the processes of feature detection, filtering, matching, and motion estimation. Each stage has its own parameter set that must be configured properly. The visual odometers under study (*libviso2* and *fovis*) share equivalent process stages and therefore they have parameters with the same meaning. It is important to compare each parameter of both odometers and extract those that are equivalent.

Table 1 enumerates the parameters for *libviso2* and *fovis* and illustrates which of them have the same meaning for both odometers, despite have different names. Furthermore, the table shows the main parameters that are unique for each algorithm, since they refer to a particular stage of the odometer.

libviso2 and *fovis* have the same basic odometry pipeline, but they differ in the underlying methods, such as the keypoint detector (Harris corners [5, 6] for *libviso2* and FAST [4, 7] for *fovis*), the feature matching (a RANSAC-based method [8, 9] for *libviso2* and a graph-based consistency algorithm [10] for *fovis*). On the other hand, some processes like feature bucketing are exactly the same for both odometers.

The total number of configurable parameters is 16 for *libviso2* and 20 for *fovis*. An overall optimization with 10 possible values for each parameter would result in an intractable amount of combinations. Thus, this work is focused on identifying key parameters (common in the algorithms under consideration) and studying their influence on the final odometry performance. In the following we give a brief overview of common parameters, contextualized by algorithm stage.

A. Keypoint detection

The features used by *libviso2* are simple blob and corner detector masks which provide a large amount of interest points. To reduce the amount of features, libviso2 applies a peak threshold technique based on finding local maximums. This technique is controlled by two main parameters: *nms-n* defines the quadratic region (window), centered around the pixel under consideration, used to compute the local maximum search. *nms-tau* defines the interest point peakiness threshold for which the pixel will be considered a keypoint or not. *fovis* uses FAST features, which are defined by the *feature-window-size* and the *fast-threshold* to decide if a feature has to be taken as keypoint or not.

B. Feature bucketing

The bucketing technique consists on divide the image into a grid and, for each rectangle, choose the best n features. *libviso2* and *fovis* implement this technique and have the same configuration parameters: *bucket-width* is the width of every bucket in the grid. *bucket-height* is the height of every bucket in the grid. *max-features* (*libviso2*) and *max-keypoints-per-bucket* (*fovis*) is the maximum number of features per bucket.

C. Stereo feature matching

In order to compute 6-DOF motion information, the 3D coordinates of the features in the left/right image pair are calculated through triangulation. *libviso2* and *fovis* use 2 parameters to control this stage: *match-disp-tolerance* (*libviso2*) and *stereo-max-dist-epipolar-line* (*fovis*) defines the maximum v-disparity tolerance. Ideally, this value would be 0, but it is adjusted to 2 or 3 in order to take into account possible errors in stereo calibration. On the other

libviso2	***fovis***	**Description**
nms-n	feature-window-size	This parameter is used in the keypoint stage to define the quadratic region centered around the pixel under consideration (the window). In the case of N=3, a window of 7x7 is used to compute the local maximum search.
nms-tau	fast-threshold	For *libviso2* this parameter is used in the keypoint state to define the threshold for a point to be considered as keypoint. For *fovis* this is the threshold for the FAST feature detector.
match-radius	target-pixels-per-feature	For *libviso2* has the same meaning as match-binsize but it is used to find matches between current and previous frames. For *fovis*, this is the reference value for the controller of the adaptive feature thresholding.
match-disp-tolerance	stereo-max-dist-epipolar-line	Defines the maximum v-disparity tolerance between matches of stereo pairs.
outlier-disp-tolerance	stereo-max-disparity	Has the same meaning than match-disp-tolerance but in the u-axis.
outlier-flow-tolerance	feature-search-window	This parameter defines the x/y-disparity for matches between current and reference frame.
refinement	use-subpixel-refinement	Specifies whether or not to refine feature matches to subpixel resolution.
max-features	max-keypoints-per-bucket	Maximal number of features per bucket. The algorithm divides the images into a grid and define a maximum number of features per bucket.
bucket-width	bucket-width	Width of bucket (in pixels).
bucket-height	bucket-height	Height of bucket (in pixels).
inlier-threshold	inlier-max-reprojection-error	Fundamental matrix inlier threshold. This parameter is used to eliminate mismatched points in the reprojection process.
match-binsize		This parameter defines the window for keypoint searching between stereo pairs.
multi-stage		If disabled, the algorithm searches matchings between current and previous left images. If it is enabled, the algorithm searches matchings between current left, current right, previous right and previous left images.
half-resolution		If enabled reduces the image by half to speed it up.
ransac-iters		Number of RANSAC iterations for the minimization of the reprojection errors.
motion-threshold		If feature flow is less than this parameter, *libviso2* algorithm replaces the reference images.
	max-pyramid-level	The maximum Gaussian pyramid level to process the images (Pyramid level 1 corresponds to the original image).
	min-pyramid-level	The minimum Gaussian pyramid level.
	fast-threshold-adaptive-gain	The threshold for the FAST detector is adaptively chosen. This parameter is the proportional factor of the controller.
	use-adaptive-threshold	True: to use the adaptive threshold defined by the parameter fast-threshold-adaptive-gain.
	ref-frame-change-threshold	If the number of inlier features is lower than this parameter, the reference frame is updated with the current images.
	use-image-normalization	If is set to TRUE, the *fovis* algorithm normalizes the image before pulling out the feature descriptors.
	clique-inlier-threshold	Threshold used in the clique algorithm to discard bad matches.
	min-features-for-estimate	Minimum number of features in the inlier set for the motion estimate to be considered valid.
	max-mean-reprojection-error	Maximum mean reprojection error over the inlier feature matches for the motion estimate to be considered valid.

Table 1: Comparison of the *libviso2* and *fovis* parameters.

hand, *outlier-disp-tolerance* (*libviso2*) and *stereo-max-disparity* (*fovis*) defines the maximum u-disparity tolerance. This value depends on the desired minimum distance from objects to be measured to camera.

D. Feature matching across time frames

The matching process across time frames uses current and reference images from left and right cameras to estimate the motion. The odometers compute feature matchings between these images in order to classify a match as a valid matching. To discard possible outliers between current and reference frames, the algorithms use the parameter *outlier-flow-tolerance* (*libviso2*) and *feature-search-window* (*fovis*) which defines the maximum x/y-disparity, in pixels, between features over the images loop.

III. PARAMETER OPTIMIZATION

From the common parameters for *libviso2* and *fovis* detailed in the previous stage, an optimization process with the following parameter set was launched (for simplicity, hereafter, we will use the parameter names of *libviso2*. Please refer to Table 1 to find the equivalences for *fovis*):

- nms-n
- nms-tau
- bucket-width
- bucket-height
- max-features
- outlier-flow-tolerance

Note that parameters of the stereo feature matching stage are not relevant for the aim of this study since the v-disparity is well defined to 2-3 for calibrated cameras and the u-disparity depends on the work environment of the odometer and can be computed with:

$$d_{max} = \frac{b \cdot f}{Z_{min}} \tag{1}$$

Where d_{max} is the maximum disparity, b is the baseline, f is the focal length and Z_{min} is the minimum distance from the camera to the objects to be considered.

A. Brute-force optimization

In order to have an overall overview of the effect of all parameters on the odometry performance, a brute-force optimization has been executed over the parameters described above. This optimization takes advantage of the use of ROS (Robotic Operating System) and the wrappers for this midleware created by our grup[1] and presented in [2].

The brute-force optimization algorithm iterates over the following steps for every parameter set:

1. Set the current parameter set.
2. Launch the odometry using a real underwater dataset.
3. Process the odometry output.
4. Save parameter set and odometry results.

We created an open source code[2] (based on Python) to launch the optimization algorithm and post-process the results.

To be consistent with the work presented previously in this matter in [11], we will follow the same odometry evaluation method. This method is based on subdividing the paths into small parts and compare velocities for each piece to the matching one in the ground truth. Following this technique, the translational and rotational errors will be taken as measures of performance of the visual odometer.

IV. RESULTS

Figure 1 shows a comparison of the influence of each parameter on the translational and rotational errors using the Spearman correlation coefficient [12]. A value close to 100% means that the parameter has a big influence on the error, while a value close to 0% means there is not influence of this parameter to the odometer performance. By examining such graphic it is clear that *max-features* has the largest impact on both the translational and rotational error. On the other hand, the bucketing size parameters (*bucket-width* and *bucket-height*) have a low impact on the traslation error, but a considerable impact on the rotational error. Refering to the parameters of keypoint detection stage (*nms-n* and *nms-tau*), they have a very low impact on the final odometry result. Finally, the *outlier-flow-tolerance* parameter has not a significant impact in the algorithms performance.

[1]See http://www.ros.org/wiki/viso2_ros and http://www.ros.org/wiki/fovis_ros

[2]See https://github.com/srv/odometry_optimization

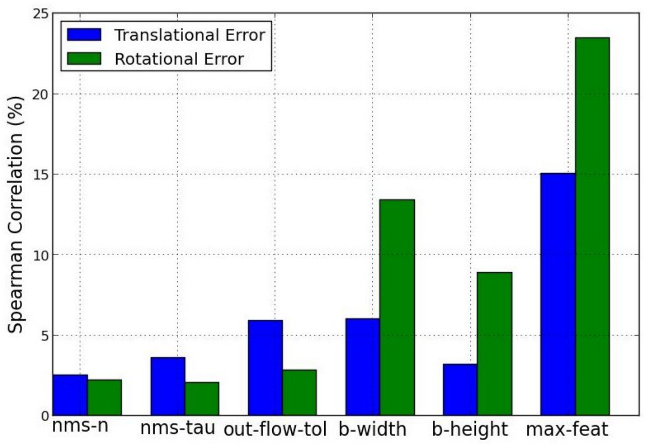

Fig. 1: Influence of each parameter on the translational and rotational errors.

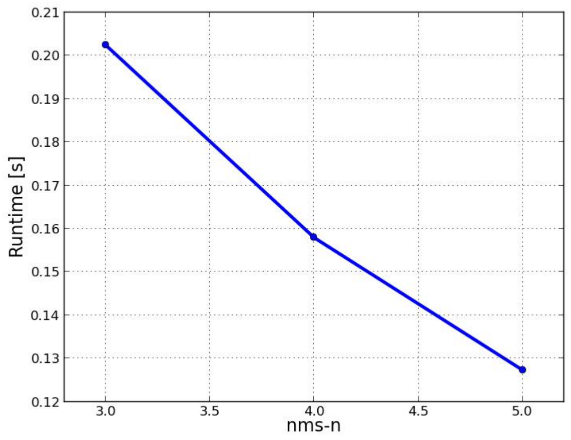

Fig. 2: Correlation between runtime and *nms-n* (Spearman coefficient of -0.72).

Another important measure of performance is the odometry runtime for every image frame. An exhaustive study using correlation techniques has been carried out to find possible relations between parameters and the algorithm runtime.

Figures 2 and 3 illustrate largest correlations found between runtime and parameters. Specifically, it is demostrated that larger window size (*nms-n*) for the keypoint detection decreases the runtime (Figure 2). This is because one feature is extracted for each window, therefore larger window size means lower number of features i.e. overall runtime decreases. Furthermore, Figure 3 shows that a larger maximum number of features per bucket (*max-features*) increases the runtime. In summary, it can be concluded that a larger number of features per image has a negative effect on the algorithm runtime.

Table 2 summarizes, qualitatively, the impact of the parameters on the translational and rotational errors and runtime.

Parameter	Tras. Error	Rot. Error	Runtime
nms-n	-	-	✓
nms-tau	-	-	-
bucket-width	-	✓	-
bucket-height	-	✓	-
max-features	✓	✓	✓
outlier-flow-tolerance	-	-	-

Table 2: Summary of the influence of the parameters on the odometry performance.

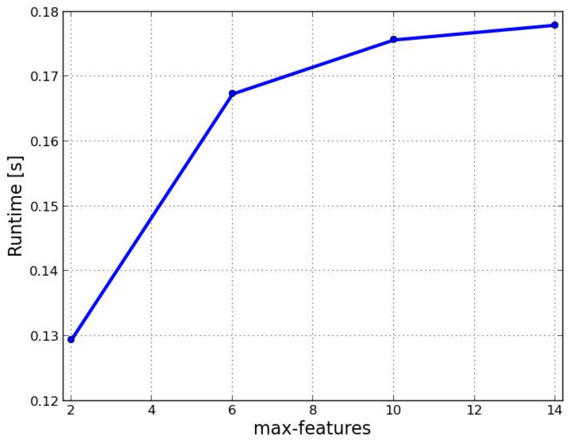

Fig. 3: Correlation between runtime and *max-features* (Spearman coefficient of 0.43).

V. CONCLUSION

In the previous work [2] it has been shown that visual odometers can be used in underwater environments in order to obtain reliable 6-DOF motion estimations. The work detailed in this paper goes a step further and performs a deep study of the parameters of two publicly available odometers (*libviso2* and *fovis*). It is focused on determine which parameters are the most important in terms of its effect on the final odometry result.

It has been shown that some parameters (such as those involved in the stereo feature matching stage) have an exact configuration that depends on the environment of the experiment to be performed and the hardware used. On the other hand, some odometry parameters have no influence on the performance, such as those related to keypoint detection stage, therefore may be left the default values.

However, some parameters have a direct impact on the odometry errors or on the execution time and must be set carefully.

REFERENCES

[1] H. Moravec, *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.* PhD thesis, Stanford University, 1980.

[2] S. Wirth, P. L. Negre, and G. Oliver, "Visual odometry for autonomous underwater vehicles," in *OCEANS 2013 IEEE International Conference*, IEEE, 2013.

[3] A. Geiger and J. Ziegler, "Stereoscan: Dense 3d reconstruction in real-time," in *IEEE Intelligent Vehicles Symposium*, jun 2011.

[4] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an rgb-d camera," in *International Symposium on Robotics Research (ISRR)*, (Flagstaff, Arizona, USA), pp. 1–16, Aug. 2011.

[5] C. Harris and M. Stephens, "A combined corner and edge detector," in *In Proc. of Fourth Alvey Vision Conference*, pp. 147–151, 1988.

[6] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry for ground vehicle applications," *Journal of Field Robotics*, vol. 23, no. 1, pp. 3–20, 2006.

[7] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European Conference on Computer Vision*, vol. 1, pp. 430–443, 2006.

[8] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 1, pp. I–652, IEEE, 2004.

[9] A. E. Johnson, S. B. Goldberg, Y. Cheng, and L. H. Matthies, "Robust and efficient stereo feature tracking for visual odometry," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 39–46, IEEE, 2008.

[10] A. Howard, "Real-time stereo visual odometry for autonomous ground vehicles," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 3946–3952, IEEE, 2008.

[11] A. Geiger, "The kitti vision benchmark suite - a project of karlsruhe institute of technology and toyota technological institute at chicago @ONLINE," Mar. 2013.

[12] A. Lehman, *JMP for Basic Univariate and Multivariate Statistics: A Step-by-step Guide.* SAS Institute, 2005.