Stereo Visual Inertial Odometry for Robots with Limited Computational Resources*

Bahnam, Stavrow ; Pfeiffer, Sven; de Croon, Guido C.H.E.

DOI
10.1109/IROS51168.2021.9636807

Publication date
2021

Document Version
Final published version

Published in
IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021

2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 27 - October 1, 2021. Prague, Czech Republic

2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-1714-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/IROS51168.2021.9636807

Stereo Visual Inertial Odometry for Robots with Limited Computational Resources*

Stavrow Bahnam[1], Sven Pfeiffer[†,1], Guido C.H.E. de Croon[1]

***Abstract*— Current existing stereo visual odometry algorithms are computationally too expensive for robots with restricted resources. Executing these algorithms on such robots leads to a low frame rate and unacceptable decay in accuracy. We modify S-MSCKF, one of the most computationally efficient stereo Visual Inertial Odometry (VIO) algorithm, to improve its speed and accuracy when tracking low numbers of features. Specifically, we implement the Inverse Lucas-Kanade (ILK) algorithm for feature tracking and stereo matching. An outlier detector based on the average sum square difference of the template and matching warp in the ILK ensures higher robustness, e.g., in the presence of brightness changes. We restrict stereo matching to slide the window only in the x-direction to further decrease the computational costs. Moreover, we limit detection of new features to the regions of interest that have too few features. The modified S-MSCKF uses half of the processing time while obtaining competitive accuracy. This allows the algorithm to run in real-time on the extremely limited Raspberry Pi Zero single-board computer.**

***Index Terms*— Aerial Systems: Perception and Autonomy, Vision-Based Navigation, Computational Efficiency**

I. Introduction

Autonomous robot navigation in GPS-denied environments is predominantly tackled with Simultaneous Localization And Mapping (SLAM) [1]. Small robots with Size, Weight and Power (SWaP) restrictions, such as lightweight Micro Air Vehicles (MAVs) typically cannot carry the sensors or processing required for SLAM. Hence, they often forego loop closure and rely on Visual Odometry (VO) to keep track of their position [2].

Monocular VO is obviously of high interest for SWaP-restricted robots, since it requires only a single camera. This saves weight, power, and also processing as only a single stream of visual inputs needs to be processed. However, it is currently still challenging to scale monocular visual odometry, where options are to add inertial measurements, i.e., to use Visual Inertial Odometry (VIO) [3], or to perform active maneuvers for stability-based scaling [4].

Stereo vision is more mature as a technology, since the known baseline distance between the cameras instantaneously adds scale. In some cases, MAVs even use two pairs of stereo cameras [5], [6] or different types of camera, such as RGB-D or infrared [7]. Unfortunately, current stereo VO algorithms require powerful [8], [9] or hardware optimized processors [10], which substantially increases the drone's cost, weight and power consumption. The computationally most restrictive platform of the above mentioned MAVs is in [9] with a dual core $1.8\ GHz$ processor. Suleiman et al. [10] achieve accurate VIO while using limited average power consumption of $24\ mW$, however, they use an application-specific integrated circuit microprocessor to run the VIO.

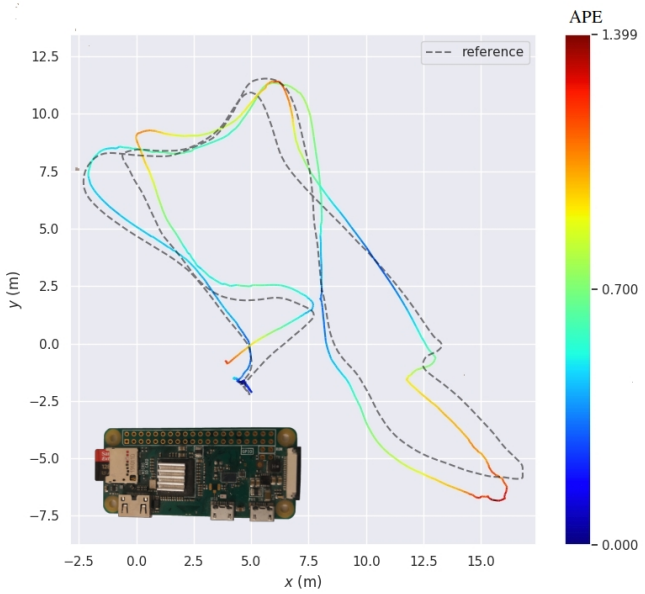


Fig. 1. We modify a state-of-the-art, efficient, stereo-based visual inertial odometry method, S-MSCKF-VIO, so that it becomes two times as efficient. The plot shows the resulting odometry and absolute point error when the modified algorithm runs real-time on a Rapberry Pi Zero (pictured on the bottom left) on the MH05 sequence from the EuRoC dataset.

The challenge addressed in this article is that of executing stereo visual odometry algorithms as computationally efficiently as possible. This will enhance the capabilities of smaller, cheaper robots, and will free up computation time on larger robots for additional tasks. As a target for very limited, lightweight, and cheap processing hardware, we employ the Raspberry Pi Zero. The RPI Zero has a size of $65\times30\times5\ mm$ and weights $9\ g$. It has an ARM1176JZF-S $1.0\ GHz$ single core processor with $512\ MB$ RAM.

Most stereo VO methods track large numbers of features in the images, both for determining motion and depth. Intuitively one might think that simply reducing the number of features is the solution to run a VO on a computationally limited device. However, the computational cost of VO algorithms does not scale well with very low number of features. This is caused by computationally expensive operations, which are constant for each frame, such as feature

*This work was supported by Royal Brinkman

[1]S. Bahnam, S. Pfeiffer and G. de Croon are with the MAVLab, Control and Operations department, Faculty of Aerospace Engineering, Delft University of Technology, Delft 2628 CD, The Netherlands (e-mail: stavrowbahnam@gmail.com; s.u.pfeiffer@tudelft.nl; g.c.h.e.decroon@tudelft.nl)

[†]Corresponding author

detection and image gradient calculations on the full image. Moreover, accuracy suffers disproportionally. Therefore, very computationally limited devices are currently not able to obtain good state estimates from stereo VO.

In this article, we present modifications to the stereo multi-state constraint Kalman filter visual inertial odometry (S-MSCKF-VIO) [11], but our contribution can be used with many other VO algorithms which track a low number of features. We have chosen S-MSCKF-VIO, because it is one of the fastest VIO and it is open source. The default version of this state-of-the-art, computationally efficient method requires a $550-600\ ms$ processing time per frame on the target platform RPI Zero. However, trying to run this algorithm real-time on the RPI0 leads to filter divergence, since too many frames are skipped. With 'real-time' we mean online processing where frame processing times longer than the frame rate results in skipping frames.

Our main contribution is that we made S-MSCKF-VIO scalable for low computational devices, making it twice as fast. To the best of our knowledge we present the first open source VIO algorithm that is able to run real-time on such a limited platform as the RPI Zero. The main modifications to S-MSCKF-VIO are:

1) Detecting features per grid cell, instead of filtering features based on their image location.
2) Using the Inverse Lucas-Kanade (ILK) [12] for feature tracking between frames.
3) Using a 1D ILK for stereo matching.

The remainder of the article is organized as follows. Firstly, in section II we discuss other work considering V(I)Os for lower computational devices. In section III the modifications of S-MSCKF-VIO are discussed. Next, the results on the EuRoC dataset are shown in section IV. This is followed by the conclusion in section V.

II. Related work

Zhao and Vela [13] compare various mono VO algorithms on different low-power devices. The device with the lowest computational power which is included in that work is the Euclid: a 64-bit embedded single-board computer system, with an Intel Atom x7-Z8700 $1.6\ GHz$ quad-core processor and $4\ GB$ of RAM [13]. The Intel Euclid developer kit has a size of $10.7 \times 19.1 \times 6.4\ cm$, but the developer kit also contains sensors such as a camera and IMU.

Delmerico and Scaramuzza [3] compare different mono VIO algorithms on computationally limited devices. The most computationally limited device they consider is the ODROID XU4, an embedded PC containing a hybrid processing unit: The Samsung Exynos5422 consists of an ARM A7 $1.5\ GHz$ quad-core and an ARM A15 $2.0\ GHz$ quad-core processor. In addition, the ODROID has $2\ GB$ of RAM, and has a size of $8.3 \times 5.8\ cm$, weighing $59\ g$. [3]

Most feature-based VO methods track features from frame to frame with the LK method. The regular LK minimizes the pixel error of the template warp from the previous frame with the warp on the new frame. However, there is a computationally more efficient algorithm, which achieves a similar tracking accuracy: the inverse compositional LK [12]. The ILK gains computational efficiency as it needs to only calculate the image gradient on a small region near the features, instead of the full image. Furthermore, it does not need to recompute the Hessian at every iteration.

The inverse compositional LK has been already used in many direct and semi-direct methods [14], [15]. However, these methods use the inverse compositional LK for pose estimation and not for feature tracking. For feature tracking, they use a computationally more expensive affine warp as opposed to using Optical Flow (OF), which is simply calculated from a sliding window. Even though the ILK can save computation time in the feature tracker, most feature-based methods use the original LK. Only the stereo VIO of Bi et al. [16] uses the ILK to track features. However, this stereo VIO is built on top of the computationally expensive ORB-SLAM2 [17] algorithm, which is too heavy for the RPI Zero. Finally, in the work of Bi et al. [16] they use the ILK to not re-compute the hessian in every iteration of the LK. In our work we will show that especially when using a low number of features, using the ILK becomes more efficient, as it will save computation time in the gradient calculation as well.

III. Method

When reducing the number of features in S-MSCKF-VIO, the feature detector ($\sim 35\ ms$) and the OF pyramid computation for LK ($\sim 80\ ms$) require the most computation of the VIO front-end. The reason for this is that both of these processes are independent of the number of features and mainly depend on the size of the image.

We therefore suggest improvements to these most time consuming tasks by employing a grid-based feature detector and the ILK. First we briefly describe the original S-MSCKF-VIO from [11] in subsection III-A. Next, we describe the grid-based feature detector in subsection III-B and lastly, in subsection III-C, we describe the implementation of the ILK for feature tracking and stereo matching.

A. S-MSCKF-VIO

S-MSCKF-VIO uses FAST9-16 [18] to detect features on the full image. After dividing the image with a grid, the features with the highest FAST-score in each grid cell are selected for stereo matching. These features are undistorted and projected to the right camera to get an initial estimation for stereo matching. A pyramidal implementation of LK from the OpenCV library is used to stereo match the features. The stereo match outliers are removed based on the distance to the epipolar line. Features are tracked between subsequent frames using the same pyramidal implementation of LK on the left image with an initial guess obtained by integrating the rotational velocities from the IMU. A 2-point RANSAC removes outliers based on the Euclidean reprojection error in the left and right image as is done in [19]. The tracked inliers are again stereo matched in the new frame.

The back-end of S-MSCKF-VIO tightly couples the IMU and VO to estimate the pose. Differently from most VIO

algorithms, the Jacobian is calculated per tracked feature and not per frame. A measurement update is performed each time a feature is lost or when the camera buffer exceeds a maximum of 20 frames. For more details one can refer to the work of Sun et al. [11]. The only change we made to the back-end was changing the maximum number of frames from 20 to 3. The reason for reducing the camera states is to reduce the size of the feature states. The second reason is to prevent adding feature observations from lost features, because this makes the computation time vary, which is not desirable.

B. Grid-based feature detection

The main idea of grid-based feature detection is to only detect new features in grid cells which have fewer features than a certain threshold. This method can reduce the average computational time of the feature detection and stereo matching of S-MSCKF-VIO, without decreasing its accuracy.

In the original S-MSCKF-VIO new features are added in the following 4 steps:

1) The features from the previous frame are tracked and a mask is created to not re-detect them.
2) FAST9-16 is used on the *full* image to detect features.
3) Per grid cell a certain maximum number of features (X) with the highest FAST-score are stereo matched.
4) The stereo matched features with the highest FAST-score are added if the number of tracked features in a cell is below a certain threshold (Y).

Step 2 and 3 detect and stereo match features on the full image, while in step 4 only features are added for grid cells which have fewer than Y tracked features. Since the location of the tracked features is already known before executing step 2, we can perform feature detection only in a Region Of Interest (ROI), which could be one or multiple cells with fewer than Y tracked features. This results in a smaller image region to detect features and therefore the algorithm has a lower average detection time. Also the number of features which need to be stereo matched is reduced, so the average stereo matching time will reduce as well. The accuracy is preserved, because the exact same features will be inserted in the unfilled cells.

The detection time using a ROI varies more between frames than without ROI, because the ROI can vary from 0 cells to the full image. However, the detection time is always equal to or smaller than without ROI. The detection time depends on the performance of the feature tracker; the higher the tracking rate, the smaller the detection time.

In an ideal case, this method essentially only detects features in new image regions due to the movement. For example, if the drone rotates to the left, the tracked features move to the right of the image, as seen in Figure 2. This results in the right-side cells being filled with tracked features, while the cells on the left-side of the image are empty and thus selected as ROI for the feature detector.

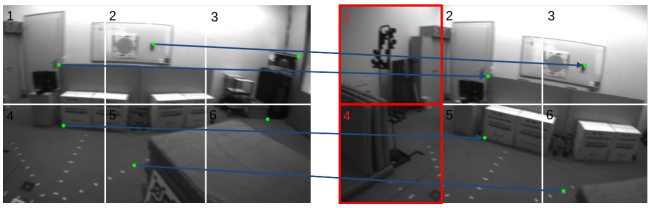


Fig. 2. Grid-based feature detection using a 2×3 grid with 1 feature per cell. If no features are lost, the ROI of the feature detector corresponds to the new image regions, shown in red (cell 1 and 4).

C. Inverse Lucas-Kanade (ILK)

The Lucas-Kanade algorithm for feature tracking tries to match a warp (window) on the new image with a template (window on previous image). It uses the gradient of the warped image and the Jacobian (derivative of the warp w.r.t. the warping parameters: 2×2 identity for a translating window) to calculate the steepest descent. For OF estimation a warp that translates a window is sufficient [12]. The algorithm is iterative, i.e., after translating the window, the gradients are recalculated for a new translation of the window, until a termination criteria is reached.

The *inverse* Lucas-Kanade algorithm saves computation time by using the image gradient of the template instead of the warped image to update the warp parameters. This means that for ILK, only the gradients of the templates are required. The computational cost of calculating the OF pyramid is thus $\mathcal{O}(nwl)$, where n is the number of features, w is the window size and l is the number of levels of the pyramid. For regular LK on the other hand, the gradient of the full image is required, which costs $\mathcal{O}(I)$, I being the original image size. It also means, that the expensive computation of the Hessian does not need to be repeated at every iteration.

We use a C++ implementation of the inverse compositional Lucas-Kanade proposed by Baker and Matthews [12] and slightly modify it. Specifically, we consider a warp that only translates a window since this is sufficient for OF estimation [12]. This allows us to further simplify the algorithm and make it computationally cheaper. The compositional part of the inverse compositional LK does not apply for translation-only warps, because the Jacobian of the warp is the 2×2 identity matrix. This also implies that the steepest descent of the image is equal to the image gradient. Furthermore, for better robustness, we use a 3×1 kernel ($I_x(x,y) = I(x+1,y) - I(x-1,y)$) instead of a 2×1 kernel ($I_x(x,y) = I(x+1,y) - I(x,y)$) to determine the image gradient.

If we assume image distortion and rotation between the cameras to be small, the warp parameters can be reduced to a single parameter for stereo matching. This corresponds to allowing the window to only translate in the x-direction after projecting the feature from the left camera to the right camera. As a result, only the x-gradient of the template needs to be computed causing the Hessian to be a scalar. Furthermore, reducing the gradient to 1D simplifies the computations to estimate the update of the warping parameter. Therefore, the ILK for stereo matching is computationally cheaper than for

TABLE I

9 DIFFERENT PARAMETER SETTINGS USED TO EVALUATE MODIFIED AND ORIGINAL S-MSCKF-VIO IN SEQUENCES V201, MH05 AND V103

Setting	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Grid geom.	2x3	2x3	2x3	3x4	3x4	3x4	4x5	4x5	4x5
Min features	1	2	3	1	2	3	1	2	3
Max features	2	3	4	2	3	4	2	3	4

feature tracking.

While the ILK has the advantage of lower computational complexity, it can have a slightly decreased performance. For instance, if the warped window arrives in a textureless area, the ILK algorithm will continue to shift in the same direction. We also noticed more brittle performance in the presence of brightness changes. For this reason, we use the average Sum Squared Difference (SSD) of the matching warp and the template warp in order to detect outliers, both for tracking and for stereo matching.

IV. RESULTS

The performance of the algorithm is evaluated on the EuRoC dataset [20], using the original WVGA images. Sequences MH01 and MH02 are excluded, because they do not start from a stationary state, as is required by S-MSCKF-VIO in order to initialize successfully. The accuracy is measured with the Root Mean Square (RMS) of the Absolute Position Error (APE), which we calculate using the tool from Grupp [21].

First we analyze the offline performance (processing all frames without taking into account the processing time) of the modified algorithm and the original S-MSCKF-VIO algorithm in subsection IV-A for different number of features. Afterwards, in subsection IV-B we show the real-time accuracy on a RPI Zero and compare it with mono VIOs running on an ODROID-XU4. Finally, we compare the offline computation time of original and modified algorithm in subsection IV-C

A. Offline VIO performance

In this section, we analyze the offline performance of the original algorithm and modified algorithm when using different numbers of features. Please note that 'offline' here means that the algorithms get sufficient time for processing a frame before a new one arrives, so that no frames are skipped. We do run the algorithms on the RPI to get representative processing times. Three EuRoC sequences are evaluated: V201 (easy, low speed), MH03 (medium, high translation speed), and V103 (difficult, high translation and rotation speed). We run the algorithms with 9 different settings, in which we vary the grid geometry, minimum number of features and maximum number of features per cell. The 9 settings can be found in Table I. All other parameters are used as in the original S-MSCKF [11], with the maximum camera states reduced from 20 to 3 for all settings. Setting 9 corresponds to the default settings with only 3 camera states. If the number of tracked features drops below the minimum in a cell, the algorithms adds new features in that cell. We also change the pyramid level from 3 to 4 for the modified algorithm, because tracking features with a big frame-to-frame displacement is harder for the ILK.

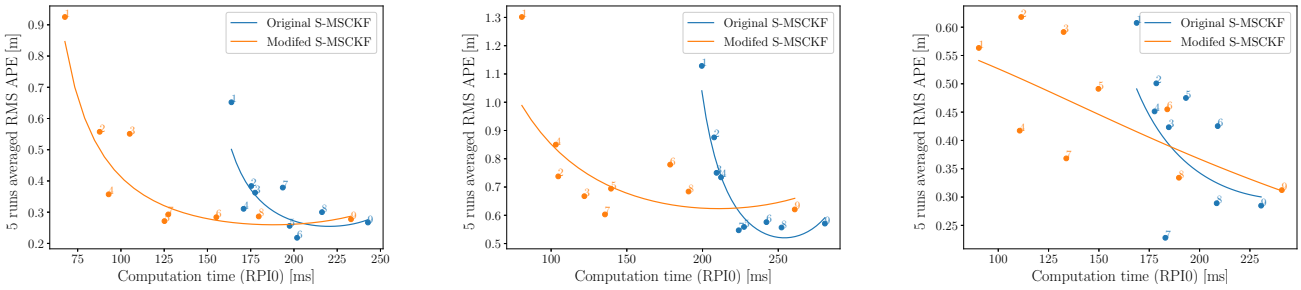
To determine the computational cost we run the algorithm on the RPI. We use a ROSbag play rate of 0.15 for settings 1-3, 0.1 for settings 4-6 and 0.05 for settings 7-9. This implies that we simulate the incoming data (images and IMU) 6.67, 10 and 20 times slower than reality respectively. We do this to ensure that we process all the frames of the sequence. Note that the computational cost of running the ROSbag is not taken into account, which accounts for about $15-20\%$ of the CPU usage on the $1\ GHz$ single core ARM processor of the RPI Zero. In order to show the trend between the RMS APE and the computation time, we plot a nonlinear fit of the form $y = a \cdot \frac{1}{x^2} + b \cdot \frac{1}{x} + c$, where y is the RMS APE and x the computation time, using all 9 data points for the original and modified S-MSCKF-VIO.

In Figure 3 the computation time vs RMS APE can be seen for sequence V201, MH03 and V103. It can be seen that the the modified S-MSCKF is faster than to the original S-MSCKF while having a similar accuracy on the three sequences. Especially for the lower settings, which correspond to lower number of features (see Table I), the computation time reduces. This is because no constant computationally expensive operations, like full image feature detection and image gradient calculations, are required for the modified algorithm. Therefore, the computation time is more sensitive on the number of features for the modified algorithm.

B. Online VIO performance

In order to run the VIO real-time on the RPI Zero, some additional adjustments have been made, because the feature tracking decayed too much resulting in a diverging filter. Specifically, new features are only detected when the grid cell has zero tracked features. This makes setting 6 the most efficient setting, since this modification especially reduces computation time for settings with higher minimum features. Because this adjustment results in some frames having a low number of features to be tracked, the performance of 2P RANSAC decreases. Therefore, we disable the 2P RANSAC and instead use a SSD threshold in the ILK function. Finally, we reduce the window size of the ILK to further decrease the computational cost.

The initialization of the VIO takes about $10-20$ seconds. Therefore, the first 3 seconds of the ROSbags are slowed down to 10% of the original rate, while the rest of the sequences is played back at full speed. The RPI Zero processes about a third of the frames (i.e. $\sim$ 7 frames per second), despite the average frame processing time being less than $100\ ms$ when running it offline. The difference is due to the increase of IMU data per frame, lower trackability of the features and additional required computing power playing the ROSbags.



(a) V201 (b) MH03 (c) V103

Fig. 3. Offline RMS APE vs computation time on a RPI0 for the original and modified S-MSCKF-VIO on sequence V201, MH03 and V103.

TABLE II

RMS APE OF VIOS RUNNING ONLINE ON COMPUTATIONALLY LIMITED DEVICES ON THE EUROC DATASET. AN 'X' INDICATES THAT FOR THAT SEQUENCE THE FILTER HAS DIVERGED, RESULTING IN A LARGE ERROR.

	Platform	***Camera***	***Alignment***	***MH03***	***MH04***	***MH05***	***V101***	***V102***	***V103***	***V201***	***V202***	***V203***
Modified S-MSCKF	RPI Zero	Stereo	SE(3)	0.61	1.31	0.71	0.25[2]	0.30	0.33	0.22	0.55	X
SVOMSF [3]	ODROID-XU4	Mono	sim(3)	0.52	2.28	1.12	0.43	0.81	X	0.15	0.46	X
MSCKF [22]	ODROID-XU4	Mono	sim(3)	0.47	0.64	0.48	0.21	0.21	1.52	0.25	0.19	1.09
OKVIS [23]	ODROID-XU4	Mono	sim(3)	X	0.42	0.62	0.09	X	X	0.11	0.26	X
ROVIO [24]	ODROID-XU4	Mono	sim(3)	0.58	0.81	0.78	0.15	0.24	0.20	0.15	0.17	0.23
VINSmono [25]	ODROID-XU4	Mono	sim(3)	0.58	0.12	0.21	0.11	0.11	0.11	0.08	0.06	0.16
SVOGTSAM [3]	ODROID-XU4	Mono	sim(3)	0.12	X	0.07	0.14	X	0.15	X	X	X

Currently, there are no stereo V(I)Os running on very computationally limited devices, however in [3] monocular VIOs are tested on an ODROID XU-4. Monocular VOs are generally computationally cheaper since they only process one image per frame. However, this comes at the cost that mono VOs cannot estimate the scale as accurately as stereo VOs. We compare our algorithm, using the the modified Pi settings, with the mono VIOs from [3] in Table II. For a fair comparison, the mono VIOs are aligned with ground truth using a sim(3) alignment, which recovers translation, rotation and scale in this post-run fit. For the modified S-MSCKF we align the trajectory with a SE(3) alignment, which only optimizes for translation and rotation. One can see that the modified S-MSCKF only has a slightly higher RMSE than the mono VIOs run on the ODROID-XU4, while it is processed on a computationally much more restricted platform and estimates the scale of the trajectory on-board.

C. Offline computation time per frame

In Figure 4 the average computation time per frame on sequence V103 can be seen. Note, that S-MSCKF runs on two threads while the RPI0 has a single core processor. Therefore, the total computation time is the sum of the two threads' computation time. We show the computation time of the original S-MSCKF with its default setting and with setting 6 from Table I. For the modified algorithm we show setting 6 and the Pi settings. Setting 6 is shown because it is most similar to the Pi settings.

The computation time of the pose optimization is reduced by reducing the maximum camera states from 20 to 3. Next, we see that the modified algorithm has a smaller feature detection time, because of the grid-based feature detector. This is even further decreased for the modified Pi settings where we only re-detect features when a grid cell is empty. Lastly, we see that the modified algorithm has a similar tracking and stereo matching time for setting 6. However, for the Pi settings the stereo matching time and feature tracking is significantly smaller. The reason for this is that the Pi settings use a window size of 9×9 instead of 15×15 in the ILK. Furthermore, the Pi settings track and stereo match fewer features as we only re-detect features when a grid cell is empty. Note that half of the (OF) image pyramid computation is added to the feature tracking time and the other half is added to the stereo matching time. The average OF pyramid calculation of the original algorithm is $75 - 80\ ms$ (all settings). For the modified algorithm only the scaled image pyramid is pre-computed (independent of the number of features) and takes $14.0\ ms$ per frame.

In Table III the average and standard deviation of the computation time per frame can be found. As expected we see that grid-based feature detection has a bigger standard deviation than the original feature detector. This is because the ROI can vary from 0 cells to the full image.

V. CONCLUSION

In this work, we made S-MSCKF-VIO a factor two more efficient by using a grid-based feature detector, implementing the ILK and restricting the ILK in x-direction for stereo matching. The increase of computational efficiency is especially present when a low number of features is used, because the ILK only needs to determine the gradient of

[2] 1 out of 5 runs failed and is excluded in the average RMSE.

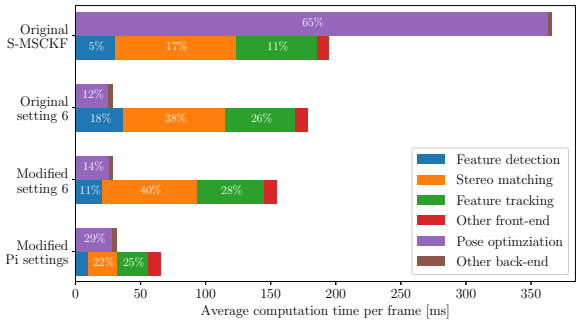


Fig. 4. Average offline processing time per image frame in ms on V103.

TABLE III

OFFLINE PROCESSING TIME PER FRAME ON SEQUENCE V103

	Feature detection (ms)		*Stereo matching (ms)*		*Feature tracking (ms)*		*Pose optimization (ms)*	
	Avg	*SD*	*Avg*	*SD*	*Avg*	*SD*	*Avg*	*SD*
Original S-MSCKF	30.5	12.1	92.9	23.6	61.8	11.1	366	343
Original setting 6	36.9	17.7	78.4	23.6	53.1	11.5	25.1	38.9
Modified setting 6	20.6	43.1	72.6	25.6	52.0	17.4	25.8	26.7
Modified Pi settings	9.9	37.4	21.8	8.4	24.3	9.4	28.1	40.6

template warps. The modified S-MSCKF runs in real-time on a Raspberry Pi Zero. The accuracy is similar to scale-recovered monocular VIOs running on a computationally much more powerful ODROID-XU4. More efficient feature management and the ILK can be applied to other algorithms, making our findings relevant not only to S-MSCKF-VIO also to other VIO algorithms that use Lucas-Kanade feature tracking and detect new features by evaluating the entire image. We believe that future work could further improve on efficiency without substantially hurting accuracy. For example, the ILK outlier detection could be performed at each step instead of only at the final iteration. Furthermore, active sampling as in [26] could be used for even more efficient feature detection.

REFERENCES

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[2] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[3] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 05 2018, pp. 2502–2509.

[4] G. C. H. E. de Croon, "Monocular distance estimation with optical flow maneuvers and efference copies: a stability-based strategy," *Bioinspiration & Biomimetics*, vol. 11, no. 1, p. 016004, jan 2016.

[5] M. G. Miiller, F. Steidle, M. J. Schuster, P. Lutz, M. Maier, S. Stoneman, T. Tomic, and W. Stürzl, "Robust visual-inertial state estimation with multiple odometries and efficient mapping on an mav with ultra-wide fov stereo vision," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 3701–3708.

[6] K. Schauwecker and A. Zell, "On-board dual-stereo-vision for the navigation of an autonomous mav," *Journal of Intelligent & Robotic Systems*, vol. 74, 04 2014.

[7] T. Mouats, N. Aouf, L. Chermak, and M. A. Richardson, "Thermal stereo odometry for uavs," *IEEE Sensors Journal*, vol. 15, no. 11, pp. 6335–6347, 2015.

[8] N. de Palézieux, T. Nägeli, and O. Hilliges, "Duo-vio: Fast, light-weight, stereo inertial odometry," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2237–2242.

[9] K. Schauwecker, N. R. Ke, S. A. Scherer, and A. Zell, "Markerless visual control of a quad-rotor micro aerial vehicle by means of on-board stereo processing," in *Autonomous Mobile Systems 2012*. Springer, 2012, pp. 11–20.

[10] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, "Navion: A fully integrated energy-efficient visual-inertial odometry accelerator for autonomous navigation of nano drones," in *2018 IEEE Symposium on VLSI Circuits*. IEEE, 2018, pp. 133–134.

[11] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. PP, 11 2017.

[12] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework part 1: The quantity approximated, the warp update rule, and the gradient descent approximation," *International Journal of Computer Vision - IJCV*, 01 2004.

[13] Y. Zhao and P. Vela, "Good feature matching: Toward accurate, robust vo/vslam with low latency," *IEEE Transactions on Robotics*, vol. PP, pp. 1–19, 01 2020.

[14] P. Liu, L. Heng, T. Sattler, A. Geiger, and M. Pollefeys, "Direct visual odometry for a fisheye-stereo camera," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1746–1752.

[15] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, "SVO: Semidirect Visual Odometry for Monocular and Multicamera Systems," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 249–265, 2017.

[16] Y. Bi, S. Lai, and B. M. Chen, "A fast stereo visual-inertial odometry for mavs," in *2018 IEEE 14th International Conference on Control and Automation (ICCA)*, 2018, pp. 265–270.

[17] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[18] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*. Springer, 2006, pp. 430–443.

[19] B. Kitt, A. Geiger, and H. Lategahn, "Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme," in *2010 IEEE Intelligent Vehicles Symposium*, 2010, pp. 486–492.

[20] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, 01 2016.

[21] M. Grupp, "evo: Python package for the evaluation of odometry and slam." https://github.com/MichaelGrupp/evo, 2017.

[22] A. Mourikis and S. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *2007 IEEE International Conference on Robotics and Automation, ICRA'07*, ser. Proceedings - IEEE International Conference on Robotics and Automation, Nov. 2007, pp. 3565–3572.

[23] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual-inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, 02 2014.

[24] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, "Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback," *The International Journal of Robotics Research*, vol. 36, pp. 1053–1072, 09 2017.

[25] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[26] G. Croon and S. Nolfi, "Act-corner: Active corner finding for optic flow determination," in *2013 IEEE International Conference on Robotics and Automation*, 05 2013, pp. 4679–4684.