

Realtime edge-based visual odometry for a monocular camera

Juan José Tarrio, Sol Pedre

Instituto Balseiro - CNEA

Bustillo 9500, Bariloche, Rio Negro, Argentina

juan.tarrio@gmail.com, sol.pedre@cab.cnea.gov.ar

Abstract

In this work we present a novel algorithm for realtime visual odometry for a monocular camera. The main idea is to develop an approach between classical feature-based visual odometry systems and modern direct dense/semi-dense methods, trying to benefit from the best attributes of both. Similar to feature-based systems, we extract information from the images, instead of working with raw image intensities as direct methods. In particular, the information extracted are the edges present in the image, while the rest of the algorithm is designed to take advantage of the structural information provided when pixels are treated as edges. Edge extraction is an efficient and highly parallelizable operation. The edge depth information extracted is dense enough to allow acceptable surface fitting, similar to modern semi-dense methods. This is a valuable attribute that feature-based odometry lacks. Experimental results show that the proposed method has similar drift than state of the art feature-based and direct methods, and is a simple algorithm that runs at realtime and can be parallelized. Finally, we have also developed an inertial aided version that successfully stabilizes an unmanned air vehicle in complex indoor environments using only a frontal camera, while running the complete solution in the embedded hardware on board the vehicle.

1. Introduction

Monocular Visual Odometry (VO) and visual SLAM have received a great deal of attention from the vision community in recent years, mainly because of its application to robot navigation, virtual reality and 3D reconstruction. Classical SLAM algorithms have relied on feature extraction and matching techniques [21] [9], creating sparse maps that are useful for trajectory tracking but not that much for world interaction. In recent years, as the growth of computational power has allowed it, direct dense methods have been developed. These algorithms operate directly on all pixels of the image, without doing any kind of data selection

Figure 1. Edge reconstruction for the fr2/desk scene. Top left picture shows the scene with edge detection and uncertainty coded in colour. Top right picture shows the edge map depth. Bottom pictures show simple surface fitting over the edge-maps for in camera and off camera view.

through feature extraction. The main advantage of these algorithms is that they generate full maps that are more useful for robot navigation and other applications. Other advantages include robustness to camera blurring and fast movement. Even though some of these algorithms are able to run in real time in modern cpu+gpu equipment [19] [18] their computational cost is still too high for embedded solutions, in which the processing power is limited by weight (in the case of light aerial vehicles), power consumption or simply by commercial price when thinking of a solution for the massive market. This led to semidense methods [6] [7] that, while applying similar techniques as fully dense, do pixel importance selection in order to reduce computational cost.

In our work we tried to find a middle point between these two approaches, mainly looking for a fast and simple algorithm to work on embedded control of Unmanned Air Vehicles, but that can also be applied to other areas of computer vision such as augmented reality and a wider branch of robotics navigation. A key idea is to give importance to the structural information provided by feature extraction, edges in particular. Edge recognition is known to be a part

human vision [17] [15], while artificial edge detectors can extract structural information in complex images [20] [1], like ultrasonic or highly blurred. From the navigation point of view, edges create similar maps as the one generated by modern semi-dense methods, as can be seen in Figure 1. The figure also shows how a surface can be easily fitted to the edge-map.

1.1. Related Work

Dense and Semi-Dense Visual Odometry: Robust odometry estimation for RGBD cameras has been reported by Kerl *et al.* [8], this work deals mainly with the problem of image alignment when operating directly on image intensities, using robust statistical approach for the optimization problem. Engel *et al.* [6] extend this work to semi-dense monocular visual odometry, adding depthmap estimation for selected pixels. Depth is estimated using variable baseline stereo comparisons and propagated over time using a probabilistic Bayesian model. This method has been used as the core to develop a full SLAM system reported in [5]. Forster *et al.* [7] reported a mixed method that combines direct image alignment with keypoints to achieve high accuracy and high framerate, intended for downward camera navigation of Micro Aerial Vehicle.

Feature Edge Based VO: Edge based SLAM has been reported by Eade and Drummond [4] and Klein and Murray [10]. The main difference with this work is that these are full SLAM systems that cluster the edge-pixels as edgels. The edgels are parametrized and treated as features, inserted into the map similar to keypoints. The main challenge of treating edges as regular features is that they are hard to match, yielding a complex data association problem, and they provide motion information only in one direction (perpendicular to the edge tangent). In our approach edges are processed densely, *i.e.* on an unclustered pixel basis. This way, our method resembles more a semi-dense visual odometry, but taking into account structural characteristics of edges.

Edge Detection: The proposed algorithm benefits from the fact that edge extraction is a highly robust, highly parallelizable and vastly researched process. To summarize, probably the most familiar edge detector is the Canny Edge Detector [2] which is based on maxima search of the gradient. Marr and Hildreth [17] proposed LoG and DoG based detectors inspired in biological vision, Lindeberg [15] followed proposing a multiscale approach. Phase congruency based detectors were proposed by Kovess [12] [11], this method presents an adimensional measure of edge strength and is robust to changes in illumination and contrast.

1.2. Method Outline

The method has 3 main steps, which are common to most classical visual odometry/SLAM systems. The first step is to process the image to extract some information, in this case structural information in the form of edge extraction. The main output of this process is an edge-map, that is, a list of edge-belonging points together with a local measure of edge direction and an estimation of the point's depth. This edge-map is the only information that is processed in the following steps. The second step is tracking. The edge-map produced from the previous frame is globally fitted into the edge-map of the new frame using an energy minimization criterion. The output of this step is a SE(3) transformation, which is represented as translation and a rotation vector. In the last step, mapping, each point in the new edge-map is matched to its corresponding point in the previous frame using the roto-translation obtained from the tracking step. Once this matching is done, a regularization step is applied on the edges. Finally, depth is updated in the new edge map using an Extended Kalman Filter (EKF) scheme. Notice that only information from the last two frames is used throughout the complete process.

2. Method Description

2.1. Edge Extraction

As expected, the choice of a particular edge detector has an impact in the final performance of the algorithm. In this particular system, robustness in terms of repetitivity is important for tracking the same edge in consecutive frames. Precise location is also important for final accuracy in velocity, position and depth estimation. Finally, low running time is another desirable quality to achieve a real time system.

A zero crossing of the DoG based detector [17] was chosen because it provides a good compromise between repetitivity and location. The first is common to DoG feature detectors (such as SIFT [16] for example), while the second is achieved by limiting the maximum value of the smoothing sigma used.

The input of the algorithm is a grayscale image which is Gaussian smoothed using two different sigmas for DoG calculation. This smoothing is efficiently performed by approximating the Gaussian with three consecutive box filters, which are calculated using integral images as suggested by [13]. The DoG is simply the difference of the smoothed images, the gradient is also calculated on the lowest sigma image.

The pixels are first thresholded using the norm of the gradient. This helps to eliminate false edges that appear on third derivative based detectors, and also discards most of the image.

For each of the remaining pixels, a third directional

derivative and subpixel position is obtained by fitting a plane to the DoG (usually using a neighbourhood of 3x3 pixels). The pixels are then thresholded using the norm of the third derivative, obtained from the coefficients of the fitted plane equation.

Also, from the plane equation, subpixel position is obtained by finding the point belonging to the line of the zero crossing of the plane that is closest to the center of the pixel. Finally, edge thinning is obtained by discarding those pixels whose subpixel position lies more than half pixel away from the center of the pixel.

Even though edge belonging pixels are not clustered into lines or any kind of group, after the edge extraction, each point on an edge is connected with the point next to it, by doing a simple search on the 8 pixels that surround each point. This joining is important for the regularization step.

To summarize, the output of the detection algorithm is a list of edge belonging points, henceforth called keylines. Notice that this keylines are 1-pixel long, so the fact that differentiates them from a keypoint is they can provide location information only in the direction of the gradient. Each keyline is parametrized as follows:

- \bar{q} : subpixel position in the image.
- \bar{m} : third derivative vector extracted from plane fitting on the DoG. This vector encodes the edge's local perpendicular direction.
- \bar{z} : estimated inverse depth, initialized at some initial value [3].
- $\bar{\sigma}$: estimated uncertainty, initialized at some big value.
- \bar{n}_d, \bar{p}_d : references to the next keylines in both directions of the tangent.

2.2. Tracking

The goal of the tracking stage is to find a SE(3) transformation that maps the previous edge-map into the new one. It is a minimization problem in which the depth of the previous edge-map and its uncertainty are taken as given parameters.

A warping function $(\bar{q}, \bar{x}) : \mathbb{R}^2 \times \mathbb{R} \times \mathbb{R}^6 \rightarrow \mathbb{R}^2$ is defined, taking as parameters keyline position \bar{q} , its estimated depth \bar{z} , and a transformation defined by the vector \bar{x} ; and returning the projected position of \bar{q} according to \bar{x} in the new image. The transformation is defined as $\bar{x} = [\bar{v} | \bar{r}]$, where \bar{v} represents a translation and \bar{r} a rotation, whose rotation matrix is obtained with its corresponding exponentiation in SO(3) as follows:

$$R(\bar{r}) = \exp(\bar{r}) \quad (1)$$

Notation Hint: Image coordinate points will be noted with letter q , and 3D points with letter p . Subindex o for “old” will be used for points in the old frame and subindex n for points in the new frame; subindex t for “transformed” will be used for points after a warping function has been applied.

Hence, the transformation for a point \bar{p}_o in space \mathbb{R}^3 is defined as:

$$\bar{p}_t = R(\bar{r}) \bar{p}_o + \bar{v} \quad (2)$$

A projection function $(\bar{p}) : \mathbb{R}^3 \rightarrow \mathbb{R}^2 \times \mathbb{R}$ that maps 3D space into image coordinates and inverse depth is also defined:

$$(\bar{p}) = \left(\frac{z_f p_x}{p_z}, \frac{z_f p_y}{p_z}, \frac{1}{p_z} \right) \quad (3)$$

$$\bar{q}^{-1}(\bar{q}, \bar{z}) = \frac{1}{z_f} \left(\frac{q_x}{z_f}, \frac{q_y}{z_f}, 1 \right) \quad (4)$$

Where \bar{p} and \bar{q} are keyline's position in 3D and image coordinates respectively, and z_f is camera focal length in pixels. The warping function then takes the form:

$$(\bar{q}, \bar{x}) = (R(\bar{r})^{-1}(\bar{q}, \bar{z}) + \bar{v}) \quad (5)$$

In order to fit the edge-maps an error measure has to be defined. In most direct methods this error takes the form of a positive definite function of the local difference of intensities. This formulation, also used in optical flow estimation, leads to an optimization problem that has to be solved pyramically to avoid falling on local minima. Nevertheless a number of well proven methods exists for solving this problem, having the advantage that since they operate directly on image intensity they don't require the feature extraction step. On the other hand, classical methods rely on feature matching in order to determine the distance between two coincident features and minimize this distance, the reprojection error.

In our case, point edges are not classical features, in the sense that they cannot be matched accurately and they only provide location information in the perpendicular direction \bar{m} ; but they are neither image intensities because some structural information is present. Hence, our criteria takes a middle point between this two approaches, trying to minimize the distance to the closest edge found after the reprojection, in the direction of the gradient \bar{m} . This is illustrated in Figure 2. Points from the previous edge-map (\bar{q}_o, \bar{z}_o) are projected into the points (\bar{q}_t, \bar{z}_t) by using the warping function. A search for the closest edge in the new image is then performed in the perpendicular direction \bar{m}_n , up to a distance of \max_d pixels. For the closest edge found, weak matching is performed by comparing the two gradients \bar{m}_o and \bar{m}_n . If the difference is above a certain

Figure 2. Edge proximity search in the tracking stage. Rotation is omitted for the sake of simplicity. Green Vectors represent edge point warping, blue vectors the search in the perpendicular direction \bar{m}_n , red dashed lines the true transformation.

threshold then no match is found and a maximum error distance \max_d is assigned to the point (*i.e.* the algorithm does not keep on searching). Otherwise, the distance error is calculated as follows:

$$d_m = (\bar{q}_t - \bar{q}_n) \cdot \bar{m}_n \quad (6)$$

Notice that the scalar product against \bar{m}_n is the projection of the distance in that direction, explicitly factoring into the error that location information is only present in this direction. The estimated variance of depth⁻² is used as a confidence parameter to weigh the error d_m , an extra weighting term w may be used also, as explained in section 2.2.1.

Putting it all together, the total error or energy to minimize takes the form:

$$E = \sum_i \frac{w_i}{2} [M((\bar{q}_{o_i}, o_i, \bar{x}) - \bar{q}_{n_i}) \cdot \bar{m}_{n_i}]^2 \quad (7)$$

$$\bar{x} = \arg \min_x E \quad (8)$$

The function $M(\cdot)$ takes the distance residual as a parameter and is defined to take into account edge matching. It returns the argument if matching is passed or \max_d distance otherwise.

2.2.1 Energy Minimization

The distance error from equation 8 varies almost linearly with the parameter vector \bar{x} , so the problem is straight forward to minimize using few iterations of classical Levenberg Marquardt and no reweighing, with satisfactory results. Nevertheless some enhancements were added to increase the agility of the system, the accuracy and the resistance to outliers.

Double Initialization: In order to help the algorithm converge in large roto-translations, the initial condition of the

optimization can be set to the estimated transformation vector in the last frame, instead of zero. The problem of this setting is that an estimation error in one frame influences negatively the convergence of the next frame. That is why a double initialization is proposed. Two iterations of optimization are performed using as initial condition the last vector and zero, separately, keeping the one with the final minimum energy.

Iterative ReWeighting: After double initialization, iterative reweighing is applied to the rest of the iterations using the square of the Hubber norm as weighing function:

$$w_i(r_i) = \begin{cases} 1 & \text{if } r_i < k \\ \frac{k^2}{r_i^2} & \text{otherwise} \end{cases} \quad (9)$$

where r_i is the distance residual and the tuning constant k is set normally at a distance of 2 pixels.

This optimization increases accuracy and also resistance to outliers. In practice, outliers are also rejected by the mapping step, so they don't get to have low variance in order to influence the solution (as explained in section 2.3).

2.2.2 Implementation details

From the point of view of computational cost, the most tricky part of minimizing equation 8 is finding the closest edge from the warped keyline. In order to accomplish this task efficiently, an auxiliary image is generated as follows: each point \bar{q}_n in the incoming edge map is propagated $\pm \max_d$ pixels in the auxiliary image along the perpendicular direction \bar{m}_n , storing only the keyline id and distance to the original edge (in integer precision). If two or more keylines are propagated to the same point in the auxiliary image, only the closest one is stored. This image is generated only once at the beginning of the optimization, and it is very fast process because it only requires an integer comparison and storing the corresponding id and distance.

Finding the closest edge at a given warped point q_t is then trivial: the id for the closest edge is located in the corresponding pixel in the auxiliary image. Usual values for \max_d are 10 pixels for 320x240 images and 20 for 640x480 images.

2.3 Mapping

Since this is an odometry system, not a full SLAM one, a global map is not generated. The only map stored is the latest edge-map. Matching is done from the new edge-map to the last, followed by a regularization step which mainly improves visualization output. Finally, depth is reestimated using standard filtering techniques.

2.3.1 Matching

In this step, the keylines in the new edge-map are matched against the ones in the last edge-map. In this manner, the chances that every point in the new edge-map has a match are increased, regardless of the possibility that two or more may point to the same keyline in the old edge-map. As suggested in [6], the warping transformation is known from the tracking stage, a stereo line is defined for each point in the image, so a more accurate matching process can be done.

Each point q_i in the new edge-map is first prerotated using $R^T(\cdot)$ (back-rotation) to obtain the transformed image points q_{r_i} :

$$\bar{r}_i = R^T(q_{i_x}, q_{i_y}, z_f)^T \quad (10)$$

$$\bar{q}_{r_i} = z_f \frac{r_{i_x}}{r_{i_z}}, z_f \frac{r_{i_y}}{r_{i_z}} \quad (11)$$

Given the linear translation $\bar{v}_r = R^T \bar{v}$ the point \bar{q}_{r_i} is related with the corresponding keyline in the last edge-map as follows:

$$q_{o_{i_x}} = q_{r_{i_x}} - o_i(z_f v_{r_x} - q_{r_{i_x}} v_{r_z}) \quad (12)$$

$$q_{o_{i_y}} = q_{r_{i_y}} - o_i(z_f v_{r_y} - q_{r_{i_y}} v_{r_z}) \quad (13)$$

As o_i is initially only known to be positive, this defines a stereo half-line to search for the match. A maximum of \max_d pixels are searched in this direction in the last edge-map image mask. When a possible match is found, gradient vectors \bar{m}_n and \bar{m}_o are compared to test for compatibility as done in the tracking stage. Because this is a weak test, o_i and o_o are used then to test for motion consistency, using the following inequality:

$$\frac{\|\bar{q}_{o_i} - \bar{q}_{r_i}\|_2}{\|(z_f v_{r_x} - q_{r_{i_x}} v_{r_z}, z_f v_{r_y} - q_{r_{i_y}} v_{r_z})\|_2} - o_i < o_i \quad (14)$$

If this last test is passed, the match is accepted and depth and uncertainty are propagated. Keylines that don't pass this test belong to objects that are mismatched or that are not moving at the same speed as the rest, *i.e.* outliers. The depth estimation of these keylines is resetted, and hence they loose their influence in the next tracking step. This provides some extra outlier motion rejection. Also, this test improves matching when the direction of the movement is very close to the edge's tangent direction, giving a number of possible matches. In this manner, only the matches that convalidate the model are kept.

2.3.2 Regularization

One regularization step is applied to the edge-map before depth reestimation. A weighted mean of the depths of the

two neighbours of each keyline and the keyline itself is performed, using the p_{id} and n_{id} obtained from the edge extractor (explained in section 2.1). Only keylines that have neighbours on both sides are regularized, excluding the end-points of edges.

The underlying idea in this regularization step is that close points in an image are likely to be close in space. This assumption is not general of course: many times the edge extractor finds two consecutive joined keylines that do not belong to the same edge in space. In order to add some resilience to this systematic error, neighbouring keylines are double tested before performing regularization. The first test is probabilistic and involves the inverse depth and its uncertainty:

$$|p - n| < p + n \quad (15)$$

The second test is morphological and uses gradient direction vector \bar{m} . A similarity function (\bar{m}_p, \bar{m}_n) is defined as the cosine of the angle between the two vectors:

$$(\bar{m}_p, \bar{m}_n) = \frac{\bar{m}_p \cdot \bar{m}_n}{\|\bar{m}_p\|_2 \|\bar{m}_n\|_2} - \cos \quad (1 - \cos)^{-1} \quad (16)$$

where θ is an angle threshold which is usually set to 45 degrees. Regularization is only performed if $\theta > 0$.

This function is used together with depth uncertainty to weigh the regularization. Having a keyline with depth and uncertainty (p, n) and its neighbours p and n , the following equations are defined:

$$w = \frac{1}{n}, w_n = \frac{(\bar{m}_p, \bar{m}_n)}{n}, w_p = \frac{(\bar{m}_p, \bar{m}_n)}{p} \quad (17)$$

The regularized depth and uncertainty for the keyline is then calculated as the weighted mean:

$$r_{reg} = \frac{w_n n + w + w_p p}{w_n + w + w_p} \quad (18)$$

$$r_{reg} = \frac{w_n n + w + w_p p}{w_n + w + w_p} \quad (19)$$

$$(20)$$

This step mainly improves visualization output, but it also helps filling some gaps when matching fails. This explains why the mean is α -weighted.

2.3.3 Depth estimation

Inverse depth α is estimated using standard EKF filtering in its multiplicative noise formulation. The prediction equation is extracted from the rototranslation calculated in the tracking stage:

$$p = \frac{o}{R(\cdot) \frac{q_{o_x}}{z_f}, \frac{q_{o_y}}{z_f}, 1, z + o v_z} \quad (21)$$

The correction equation is similar to equations 12 and 13 but rearranged for a forward rotation and, most importantly, scalar multiplied by the vector \bar{m}_n , the only direction in which the edge provides motion information:

$$\begin{pmatrix} q_{n_x} - q_{r_x} \\ q_{n_y} - q_{r_y} \end{pmatrix} \cdot \bar{m}_n = \frac{1}{p} \begin{pmatrix} z_f v_x - q_{r_x} v_z \\ z_f v_y - q_{r_y} v_z \end{pmatrix} \cdot \bar{m}_n \quad (22)$$

where q_r is the image point that results from rotating q_o by $R(\cdot)$.

Error Estimation: Uncertainty is estimated in the EKF by a first-order approximation around the filter operating point. In this stage, velocity and rotation appear as parameters. But, as they are estimated quantities, they have some uncertainty that is taken into account in the filter formulation. This error term is extracted from the Levenberg-Marquardt minimization performed at the tracking stage. In this manner, uncertainty is modelled as the sum of the covariance matrix extracted from the last iteration of the algorithm $((J_n^T J_n)^{-1})$ plus a relative error term derived from the confidence interval: $\frac{\|h_n\|}{\|\bar{x}\|}$, being h_n the parameter increment in the last iteration and $\bar{x} = [\bar{v}]$. Hence, the modelled uncertainty in the transformation parameters takes the form:

$$R_{\bar{x}} = (J_n^T J_n)^{-1} + \text{diag}(\bar{x})^2 \quad (23)$$

Modelling this uncertainty introduces the insight that velocity and rotation are not accurate at the beginning because the reconstruction is still uninitialized. After the system is initialized, the main source of uncertainty is the localization error, usually taken to be 1 pixel.

This two uncertainty sources are combined using the standard filter equations. Uncertainty in the state propagation is also added with a constant term, designed to keep the estimation active and avoid that some keylines may get fixed at wrong positions at the beginning of the estimation.

Global scale correction: Scale drift is a known problem of visual odometry systems that has an impact on the final accuracy of the trajectory. One of the causes of scale drift in this system is the presence of a multiplicative noise term in the EKF, which appears because of the velocity term that multiplies in equations 22. This introduces only a small bias in the estimation of \bar{v} , because the variance of the velocity is much less than the variance of the keyline's displacement measurement. Nevertheless, explicit bias correction is hard to apply because the involved variances need to be known with precision.

Taking as an advantage the great number of points that are processed in each frame, a “shrinking factor” can be estimated by comparing the predicted EKF inverse depth p

with the final depth after measurement incorporation, globally:

$$k = \frac{\frac{1}{2} \frac{1}{p_i}}{\frac{1}{2} \frac{1}{p_i}} \quad (24)$$

This factor is used to expand the system back, by dividing each depth by this factor.

The presented correction step is not required for the system to work, it is just a simple correction to gain accuracy.

3. Inertial Aided Formulation

Aside from the pure vision algorithm presented so far, we have also developed an inertial aided system that combines the vision process with inertial measurements (gyroscope plus accelerometer) to effectively stabilize and navigate a Vertical Take Off and Landing UAV. Although the vision system could be enough for this task, the reasons to include inertial information for this application are several.

For one, the rotation information given by the gyroscope is fundamental for fast low-level rotation control of this kind of vehicle. Although visual rotation control could be feasible, it's complexity is not justified by the low cost of this kind of sensors. Moreover, adding gyroscope information to the visual system increases its robustness in terms of convergence and motion range while lowering computational cost. Finally, the pure monocular system is unable to estimate the world scale. Although this information can be retrieved with an initialization step, it can also be derived from accelerometer measures, with no initialization or external information needed.

Inertial information is added to the visual estimation by first estimating a measured rotation matrix $R_m(\bar{\omega}_m)$ from the angular velocity output of the gyroscope. This rotation matrix is used to prerotate the old edge-map before the tracking stage. The estimated rotation is mixed with the measured one as two consecutive rotations: $R = R_e R_m$. After mixing the algorithm continues as normal.

A rotation prior can be added to the energy minimization in the form of a time regularization term:

$$E = E_o + \lambda (\bar{\omega}_t - \bar{\omega}_{t-1}) \quad (25)$$

In this equation λ is a regularization factor. Choosing a big λ converts rotation estimation into a gyroscope bias estimator.

In the most minimalistic setup, the energy function is only minimized with respect to translation, while rotation is extracted from the Gyro. This last configuration is faster to compute than the complete version and provides satisfactory results for a well-calibrated sensor (as shown in section 4.3).

Dataset	Pos. Drift [cm/s]				Ang. Drift [deg/s]			
	REBVO	[6]	[8]	[9]	REBVO	[6]	[8]	[9]
fr2/xyz	0.8	0.6	0.6	8.2	0.40	0.33	0.34	3.27
fr2/desk	2.8	2.1	2.0	-	1.07	0.65	0.70	-

Table 1. Drift comparison against state of the art methods.

4. Results

The algorithm was tested using the TUM RGBD public dataset [22] and datasets generated by the authors, using the WHYCON system [14] for position ground truth.

4.1. Experimental Setup

The tests were conducted using two platforms: a regular desktop featuring an Intel I7 core and an embedded platform consisting of an Odroid U3 development board featuring an Exynos 4460 SoC (Cortex-A9 Quad). The Odroid U3 has an inertial measurement unit plugged in and a PlayStation 3 Eye camera. This equipment is mounted on top of a quadrotor, and is the one used to create the second set of measurements.

Initialization is a critical part for monocular systems. In all the experiments we left the system auto-initialize, starting all the keylines with a fixed constant depth, this way the edge-map takes a few seconds (2-5) to converge. Because absolute scale is unknown in monocular systems, this was adjusted offline by comparing against ground truth.

4.2. Tests on the TUM RGBD Dataset

To facilitate comparison with other visual odometry systems, the algorithm was tested in a public dataset. Two sequences were used: fr2/xyz and fr2/desk, both have been used in related work to quantitatively assess the performance of VO algorithms. The first one contains very slow translational motions around a regular office desk. The second one is a complete loop around this desk with faster motions. These datasets use the camera from a Kinect sensor and a ground truth positioning system. The algorithm was initialized using the first depth map taken by the Kinect.

Table 1 shows drift information compared against state of the art visual odometry systems. As can be seen, the proposed algorithm presents similar positional and angular drift. Note that only [9] and [6] are monocular systems. For these experiments, the first one requires stereo alignment for initialization and the former used the depth image from the first frame, provided by the RGBD dataset, for the same purpose. Our system auto-initializes.

Reconstruction for the last frame of the dataset is shown in Figure 1.

Figure 3. Edge reconstruction in a typical hallway with low amount of distinguishable features. From left to right: edge detection, edge depth and off-camera surface fitting.

4.3. Tests on generated datasets

Even though testing using public datasets is important for comparison with other methods, it is also important to test the algorithm in the intended environment and platform for at least one of its applications (small UAV stabilization and navigation). Figure 3 shows the output of the algorithm for a typical office corridor and its respective coloured edge-map, it also shows surface fitting over the edge map. Figure 4 shows position tracking of the vehicle in a 40s trial on this environment. It also shows the performance of the algorithm when the rotation is extracted from the gyroscope and the algorithm is only used to estimate translational velocity (no estimation of rotation bias). As expected, this estimation has more position drift, but runtime is far lower as can be seen in section 4.4, yielding a tradeoff between speed and accuracy which is common in embedded systems.

Velocity estimation is plotted in Figure 5 (for the pure visual estimator), and angular velocity is plotted in Figure 6 against gyroscope measurements (which is an excellent sensor for angular velocity). These graphics are mainly plotted to show the lack of noise (in terms of smoothing) in the velocity estimations and the precision of the system. A

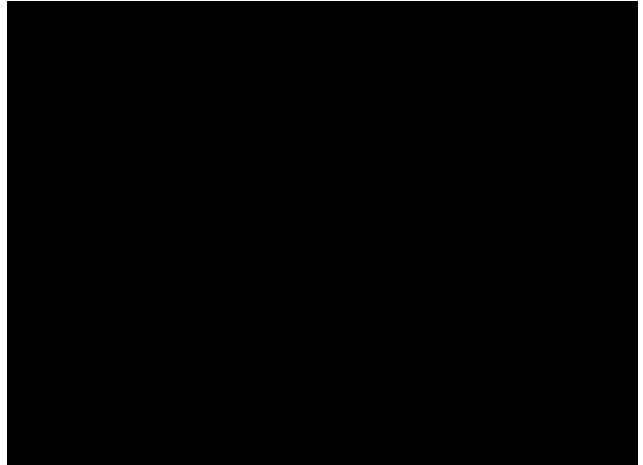


Figure 4. Position estimation versus WHYCON ground truth for the hallway dataset. An initialization time of 5 seconds is taken as the initial position point in the plots.

typical office reconstruction is shown in Figure 7, note how the algorithm is able to track the line of trees and this way

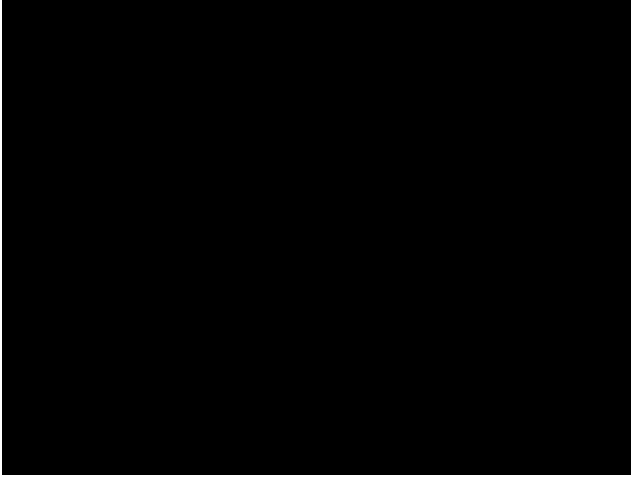


Figure 5. Velocity estimation versus WHYCON ground truth for the hallway dataset.

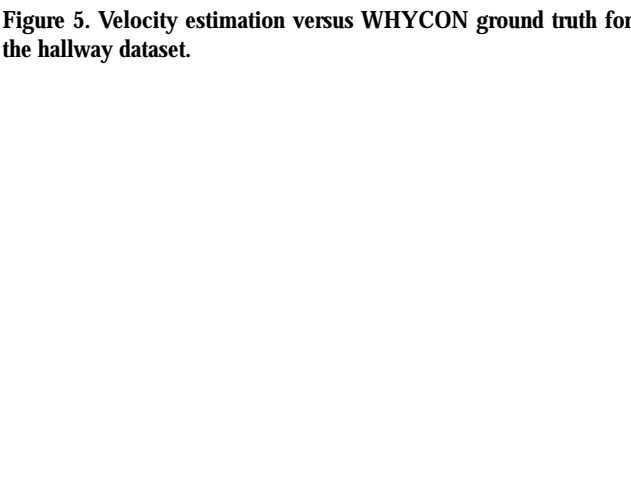


Figure 6. Angular velocity estimation versus on-board gyroscope measurements for the hallway dataset.

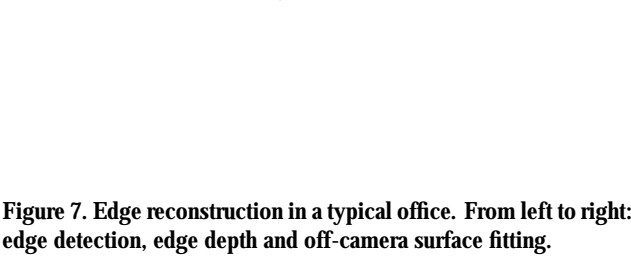


Figure 7. Edge reconstruction in a typical office. From left to right: edge detection, edge depth and off-camera surface fitting.

identify the window.

4.4. Running Time

Running time is of key importance for a realtime system designed to work in a control loop. One of the attractive things of this algorithm is that the running time for both mapping and tracking scales linearly with the number of edge-points processed, as can be seen in Figure 8. Therefore it can be accurately controlled actuating on the thresholds of

	Runtime [ms]	
	Odroid U3	I7
REBVO	55	8
REBVO+GIRO	26	4

Table 2. Average running time for different platforms and configurations.

the edge detector. Average running times for the algorithm are shown in Table 2 for 4500 edge-points. The gyroscope version runs faster because the algorithm only estimates velocity, this simplifies the optimization and requires fewer iterations to converge.

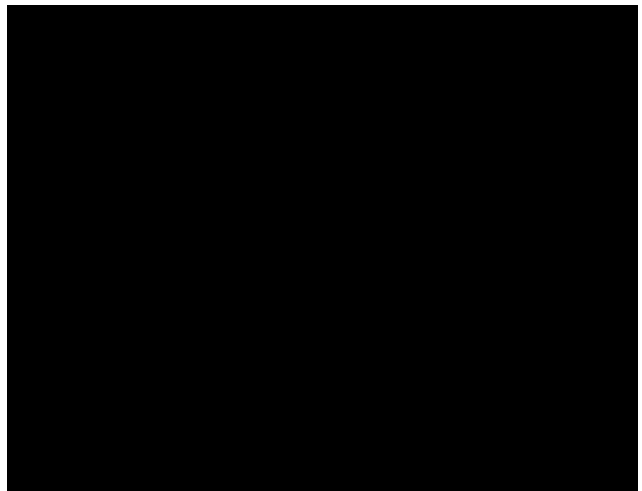


Figure 8. Runtime as a function of edge-points on a Cortex-A9

5. Conclusion

In this paper, a novel visual odometry algorithm that takes an approach between classical feature-based visual odometry and novel semi-dense direct methods has been presented. While mainly aiming for simplicity and efficiency on embedded platforms, the proposed algorithm achieves similar accuracy than modern state of the art visual odometry systems. In terms of running time, the algorithm in its minimalistic version using gyroscope is able to run at 30FPS on a standard Cortex-A9 without mayor optimizations to the code (like the use of explicit vectorization, which can be easily applied in the future). The full version runs at 18FPS on a Cortex A9 and 120 FPS in a modern CPU. Another important characteristic is that the runtime can be pitched by thresholding the number of edge points, following a simple linear relation. The depth output could also be used with a system similar to the proposed in [5] to achieve a full SLAM system.

References

- [1] A. Belaid, D. Boukerroui, Y. Maingourd, and J.-F. Leralut. Phase-based level set segmentation of ultrasound images. *Information Technology in Biomedicine, IEEE Transactions on*, 15(1):138–147, 2011. 2
- [2] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1986. 2
- [3] J. Civera, A. J. Davison, and J. Montiel. Inverse depth parametrization for monocular slam. *Robotics, IEEE Transactions on*, 24(5):932–945, 2008. 3
- [4] E. Eade and T. Drummond. Edge landmarks in monocular slam. In *BMVC*, pages 7–16, 2006. 2
- [5] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014*, pages 834–849. Springer, 2014. 2, 8
- [6] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1449–1456. IEEE, 2013. 1, 2, 5, 7
- [7] C. Forster, M. Pizzoli, and D. Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22. IEEE, 2014. 1, 2
- [8] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for rgb-d cameras. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3748–3754. IEEE, 2013. 2, 7
- [9] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007. 1, 7
- [10] G. Klein and D. Murray. Improving the agility of keyframe-based slam. In *Computer Vision–ECCV 2008*, pages 802–815. Springer, 2008. 2
- [11] P. Kovesi. Image features from phase congruency. *Videre: Journal of computer vision research*, 1(3):1–26, 1999. 2
- [12] P. Kovesi. Edges are not just steps. In *In: Proceedings of the Fifth Asian Conference on Computer Vision. (2002) 822827*, pages 822–827, 2002. 2
- [13] P. Kovesi. Fast almost-gaussian filtering. In *Digital Image Computing: Techniques and Applications (DICTA), 2010 International Conference on*, pages 121–125. IEEE, 2010. 2
- [14] T. Krajnk, M. Nitsche, J. Faigl, P. Vank, M. Saska, L. Peuil, T. Duckett, and M. Mejail. A practical multirobot localization system. *Journal of Intelligent Robotic Systems*, pages 1–24, 2014. 7
- [15] T. Lindeberg. Principles for automatic scale selection. *Handbook on Computer Vision and Applications*, 1999. 2
- [16] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. 2
- [17] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London B: Biological Sciences*, 207(1167):187–217, 1980. 2
- [18] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE, 2011. 1
- [19] M. Pizzoli, C. Forster, and D. Scaramuzza. Remode: Probabilistic, monocular dense reconstruction in real time. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2609–2616. IEEE, 2014. 1
- [20] K. Rajpoot, V. Grau, and J. A. Noble. Local-phase based 3d boundary detection using monogenic signal and its application to real-time 3-d echocardiography images. In *Biomedical Imaging: From Nano to Macro, 2009. ISBI'09. IEEE International Symposium on*, pages 783–786. IEEE, 2009. 2
- [21] H. Strasdat, J. Montiel, and A. J. Davison. Real-time monocular slam: Why filter? In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2657–2664. IEEE, 2010. 1
- [22] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012. 7