Visual-inertial Tracking on Android for Augmented Reality Applications

Lorenzo Porzi, Elisa Ricci and Thomas A.Ciarfuglia
Dipartimento di Ingegneria Elettronica e dell'Informazione
University of Perugia, Italy
Email: info@lorenzoporzi.com, [ricci, ciarfuglia]@diei.unipg.it

Michele Zanin
Fondazione Bruno Kessler
Povo (TN), Italy
Email: mizanin@fbk.eu

Abstract— **Augmented Reality (AR) aims to enhance a person's vision of the real world with useful information about the surrounding environment. Amongst all the possible applications, AR systems can be very useful as visualization tools for structural and environmental monitoring. While the large majority of AR systems run on a laptop or on a head-mounted device, the advent of smartphones have created new opportunities. One of the most important functionality of an AR system is the ability of the device to self localize. This can be achieved through visual odometry, a very challenging task for smartphone. Indeed, on most of the available smartphone AR applications, self localization is achieved through GPS and/or inertial sensors. Hence, developing an AR system on a mobile phone also poses new challenges due to the limited amount of computational resources. In this paper we describe the development of a egomotion estimation algorithm for an Android smartphone. We also present an approach based on an Extended Kalman Filter for improving localization accuracy integrating the information from inertial sensors. The implemented solution achieves a localization accuracy comparable to the PC implementation while running on an Android device.**

I. Introduction

Currently, many monitoring systems rather than being based on a fixed configuration of sensors tend to make use of mobile sensors to collect information (see e.g. [1], which refers to mobile robotics for environmental monitoring). A key issue for such systems, whatever the specific application, is the ability to self localize, in order to associate the information gathered with the position of the sensing device and/or that of the measured spot. On the other hand, often monitoring tasks have to be performed on site by human operators, and in these cases the availability of hand-held devices is fundamental. Augmented Reality (AR) may play a major role in the realization of a hand-held monitoring device, as the visualization of the data collected from various sources and of the information retrieved through some specific kind of processing, can be integrated in the current view provided by the device. In [2] an example of these applications is presented, demonstrating an AR system to support geoscientists in environmental monitoring. In [3] an AR visualization tool for structural engineers is proposed aimed on monitoring the health of a bridge structure. In [4] a prototype AR interface is designed for indoor environmental monitoring. The processed and visualized data are the sound and temperature data which are located inside a networked environment.

In this scenario smartphones, providing advanced PC-like functionality, constitute universal platforms that can be used to develop complex AR applications in various domains. In this work we present the development of a localization algorithm for Android smartphones by porting the Parallel Tracking and Mapping (PTAM) algorithm [5]. While most of current AR systems are implemented on laptops or ad-hoc devices, and use markers to identify objects of interest, we aim at developing an hand-held system which is able to localize itself from visual data and learn a map of the surrounding environment, thus avoiding the use of markers.

PTAM has already been adapted by its authors to work on a smartphone. This adaption was targeted to the iPhone 3G, a very low-power device for today standards, that posed severe limitations to the complexity of the algorithm, consistently reducing pose estimation accuracy. Additionally, the source code of this porting is not publicly available. Recent smartphones offer both improved processing power and a wider array of integrated sensors making it possible to work on a closer adaption of the original algorithm, as well as the integration of sensor fusion into the system. These improved capabilities however are still not enough for real-time operation of a direct porting of the original desktop version of PTAM. In the following we describe the development of an efficient version of this algorithm for Android smartphones, featuring also a sensor fusion module that processes the data available from the Inertial Measurement Unit (IMU) of the smartphone (measurements provided by an accelerometer and a gyroscope) to enhance the quality of localization.

II. Parallel Tracking and Mapping

PTAM is a monocular egomotion estimation algorithm aimed at tracking an hand-held camera in a small, unknown, mostly static environment. In PTAM the typical SLAM (Simultaneous Localization and Mapping) tasks of camera tracking and map making are decoupled and run in two parallel threads. After an initial map of 3D points and keyframes is built using a stereo initialization procedure, the Mapping thread uses bundle adjustment to optimize it. In the meanwhile the Tracking thread estimates the camera pose using feature matches between 3D map points and 2D corners found in the video frames. As the camera explores the scene the map is grown by adding keyframes and map points [5]. In the

978-1-4673-2738-1/12/$31.00 ©2012 IEEE

following we focus on the tracking thread and its adaption for the Android OS.

The tracking thread receives as input the video frames captured by the camera and uses the structural information contained in the map to estimate the camera pose relative to the world coordinate frame established during map initialization.

To proceed further we introduce some notations. We refer to the world coordinate frame as $\mathcal{W}$ and to the coordinate frame fixed with the camera as $\mathcal{C}$. The coordinates system in which a vector is expressed is denoted by a superscript, e.g. $\mathbf{v}^{\mathcal{W}}$ is the vector $\mathbf{v}$ expressed in the world coordinate frame $\mathcal{W}$. The camera pose $E_{\mathcal{W}}^{\mathcal{C}} = (\mathbf{q}_{\mathcal{W}}^{\mathcal{C}}, \mathbf{t}_{\mathcal{W}}^{\mathcal{C}})$ is the affine transform that brings points from the $\mathcal{W}$ coordinate frame to the $\mathcal{C}$ coordinate frame, parametrized with the unit quaternion $\mathbf{q}_{\mathcal{W}}^{\mathcal{C}}$, and the vector $\mathbf{t}_{\mathcal{W}}^{\mathcal{C}}$, expressing, respectively, the rotation from frame $\mathcal{W}$ to frame $\mathcal{C}$, and the origin of the frame $\mathcal{W}$ expressed in the frame $\mathcal{C}$. We write $\mathbf{v}^{\mathbf{C}} = E_{\mathcal{W}}^{\mathcal{C}} \mathbf{v}^{\mathbf{W}}$ as a shorthand notation for $\mathbf{v}^{\mathbf{C}} = \mathbf{t}_{\mathcal{W}}^{\mathcal{C}} + \mathbf{q}_{\mathcal{W}}^{\mathcal{C}} \mathbf{v}^{\mathbf{W}}$. We remark that as any monocular SLAM system algorithm, PTAM provides estimates of the unscaled position, hence $\mathbf{t}_{\mathcal{W}}^{\mathcal{C}}$ represents the position coordinates of the camera in the arbitrary scale established during map initialization.

The functioning of the tracking thread is summarized in Figure 1. When a new frame is received, the Pose Prediction module calculates an a-priori estimate of the camera pose using a simple decaying velocity model similar to an alpha-beta filter [8], which maintains an estimate of the camera velocity $\hat{v}_k$. At frame k, given the pose of the previous frame $E_{\mathcal{W}}^{\mathcal{C}_{k-1}}$ an a priori pose is calculated as:

$$\tilde{E}_{\mathcal{W}}^{\mathcal{C}_k} = \exp(\hat{v}_k) E_{\mathcal{W}}^{\mathcal{C}_{k-1}} \tag{1}$$

The a priori pose $\tilde{E}_{\mathcal{W}}^{\mathcal{C}_k}$ is refined in subsequent steps of the tracking thread:

1) The FAST [9] corner detection algorithm is used to find prominent corners in the image.
2) A small subset of map points is projected on the current frame using the a priori pose.
3) The map points are compared to the corners lying close to their projection and a set of 3D to 2D correspondences is found.
4) The correspondences are used to refine the pose estimate using a robust re-projection error minimization algorithm.
5) The last three steps are repeated, this time using a larger set of map points and the pose calculated in the previous iteration (coarse to fine refinement).

After frame tracking is completed, the refined pose $E_{\mathcal{W}}^{\mathcal{C}_k}$ is used to update the motion model:

$$\hat{v}_{k+1} = \alpha \frac{\hat{v}_k + \log(E_{\mathcal{W}}^{\mathcal{C}_k} (E_{\mathcal{W}}^{\mathcal{C}_{k-1}})^{-1})}{2} \tag{2}$$

where $\alpha < 1$ is a velocity damping factor. In the previous formulas $\exp(\cdot)$ and $\log(\cdot)$ are respectively the exponential map and its inverse [10].

III. PTAM implementation for Android smartphones

Our adaptation of PTAM for mobile devices brings some important modifications to the original algorithm. Corner detection and coarse to fine refinement are eschewed in favor of an exhaustive corner searching method, similar to the one used in [11]. These changes are motivated by the results obtained during a preliminary testing phase documented in section V-A. The revised tracking algorithm is summarized in Figure 2.

The choice of Android [7] as the target platform is the source of some interesting engineering challenges. Android is designed to run on multiple devices and processor architectures and its software development process is strongly oriented towards portability. To achieve this portability goal every user application and a large part of the system libraries are written in the Java programming language and compiled to bytecode that is executed at run time on a virtual machine, called Dalvik VM. The use of native C/C++ code for low level optimizations is nevertheless supported by the Android SDK but requires some additional work on the part of the developer. In particular a wrapper layer must be written to manage data exchange between managed code running inside the VM and native unmanaged code running outside the VM. This wrapper is the source of a considerable performance overhead that, coupled with the support of JIT compilation by the Dalvik VM, renders the use of native code convenient from a performance perspective only if it can bring quite big optimizations. Indeed these optimizations are possible, particularly if the algorithm can take advantage of architecture-specific instructions (e.g. vectorial instructions).

In our adaptation of PTAM these optimization issues are addressed following a two-fold approach: as much of the algorithm as possible is implemented with native code to minimize data interchange between the managed and the unmanaged parts of the program, and algebraic computations are performed using the Eigen library [6]. This C++ template library provides a set of common algebraic data structures and algorithms and supports automatic vectorization taking advantage of vectorial instructions on several CPU architectures. This last feature in particular is very useful and has a big impact on overall performance: as a reference, Figure 3 shows computing times for the same simple vector summation method written in Java, in C++ without vectorial instructions, in C++ with ARM NEON instructions used manually and in C++ using Eigen.

IV. Sensor Fusion

Most modern smartphones have several sensors on board, hence sensor fusion is quite a natural candidate to complement a system like ours. Fusing visual and inertial data for ego-motion estimation is an established topic (see e.g. [12], [13]). These approaches usually involve the use of more accurate and faster sensors than the ones found in smartphones, so adapting them to our case poses an additional challenge. As mentioned above, the PTAM provides estimates of the unscaled position, together with the attitude of the camera. In such cases, often

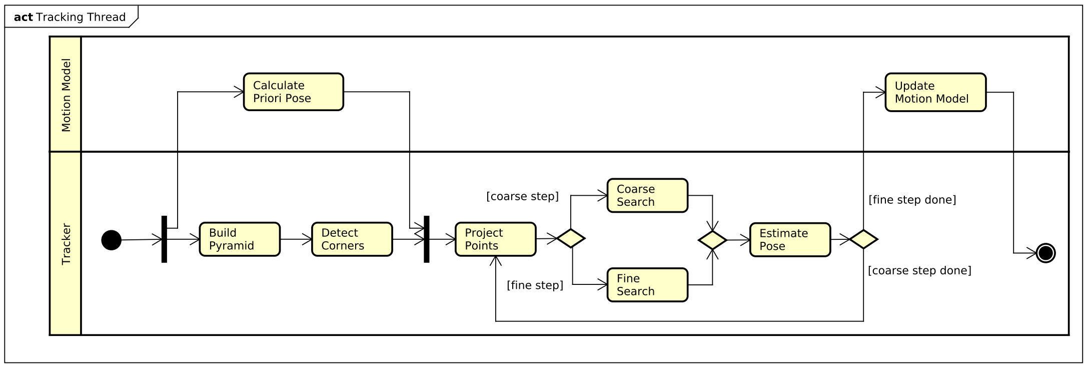

Fig. 1. Activity diagram of PTAM's tracking thread.

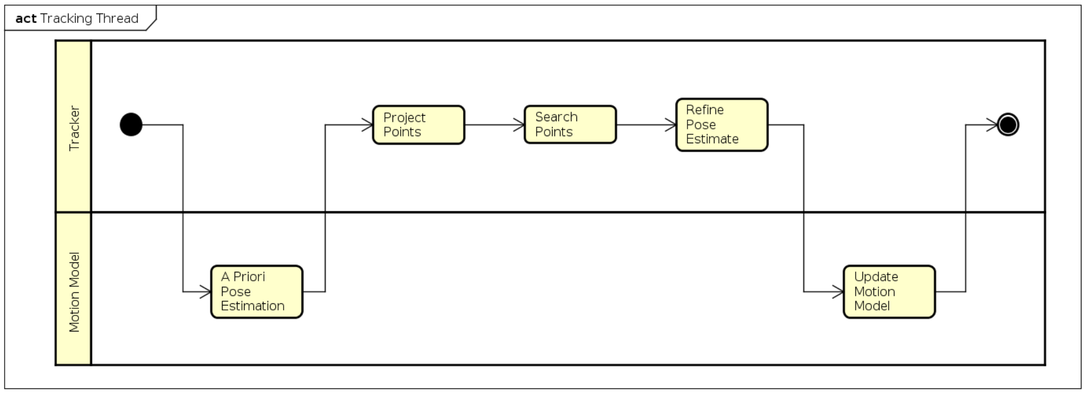

Fig. 2. Activity diagram of the adapted tracking thread.

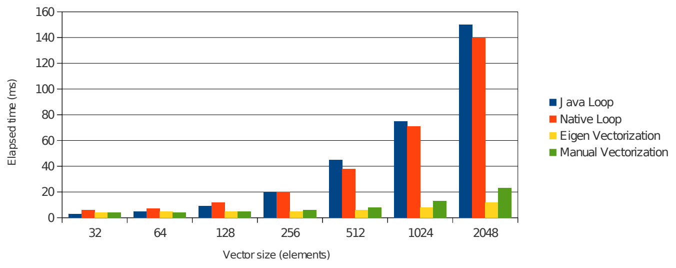

Fig. 3. Running times for 1024 repetitions of vector sum.

the sensor fusion module is devoted to estimate the scale factor, resulting in a sort of on-the-fly calibration (see e.g., [14]). Here, we describe a sensor fusion module based on Extended Kalman Filter (EKF) aimed rather at smoothing the trajectory estimates given by the PTAM, and possibly at substituting the Pose Prediction module of the PTAM. While sensor fusion [20] and the use of EKF is a well established topic (see e.g. [18], [19]), we are not aware of methods specifically proposed for operating on Android devices.

Accurately modeling the dynamics of an hand-held camera is not feasible in our context: there are simply too many variables to take into account. While the pose of a camera has always six degrees of freedom, be it hand-held or, for example, mounted on a wheeled vehicle, the physics of the vehicle constrain in an obvious way the camera's motion, while the constraints given by the human arm–hand system are not obvious at all. For these reasons, the dynamic model commonly adopted in literature (see e.g., [14], [15], [16], [17]), is constituted by a double integrator of the accelerations, on the dynamics of the rotation matrices, and on the estimation of the (constant) scale factor and gravity acceleration. The state vector is defined as $\mathbf{x} = [\mathbf{t}^{\mathcal{W}}, \mathbf{v}^{\mathcal{W}}, \mathbf{a}^{\mathcal{W}}, \mathbf{q}_{\mathcal{C}}^{\mathcal{W}}, \omega^{\mathcal{W}}, \mathbf{g}^{\mathcal{W}}, \lambda]^\top$, with $\mathbf{t}^{\mathcal{W}}, \mathbf{v}^{\mathcal{W}}, \mathbf{a}^{\mathcal{W}}$ representing respectively unscaled position, velocity and acceleration of the center of the camera frame, $\mathbf{q}_{\mathcal{C}}^{\mathcal{W}}, \omega^{\mathcal{W}}$ are the rotation from camera frame to world frame and relative angular velocity, and $\mathbf{g}^{\mathcal{W}}, \lambda$ the (constant) gravitational acceleration and the scale factor. The continuous time model is:

$$\begin{cases} \dot{\mathbf{t}}^{\mathcal{W}}(t) = \lambda(t)\mathbf{v}^{\mathcal{W}}(t) \\ \dot{\mathbf{v}}^{\mathcal{W}}(t) = \mathbf{a}^{\mathcal{W}}(t) \\ \dot{\mathbf{a}}^{\mathcal{W}}(t) = \nu_a(t) \\ \dot{\mathbf{q}}_{\mathcal{C}}^{\mathcal{W}}(t) = \frac{1}{2}\begin{bmatrix} \omega^{\mathcal{W}}(t) \\ 0 \end{bmatrix} \mathbf{q}_{\mathcal{C}}^{\mathcal{W}}(t) \\ \dot{\omega}^{\mathcal{W}}(t) = \nu_\omega(t) \\ \dot{\mathbf{g}}^{\mathcal{W}}(t) = 0 \\ \dot{\lambda} = 0 \end{cases}$$

while a discrete-time model can be obtained by ordinary first order linearization and by using the exponential map for the integration of the quaternion dynamics, see [10], in the sampling interval $\triangle t$:

$$\begin{cases} \mathbf{t}^{\mathcal{W}}(t) = \mathbf{t}^{\mathcal{W}}(t-\triangle t) + \triangle t\ \lambda(t-\triangle t)\mathbf{v}^{\mathcal{W}}(t-\triangle t) \\ \mathbf{v}^{\mathcal{W}}(t) = \mathbf{v}^{\mathcal{W}}(t-\triangle t) + \triangle t\ \mathbf{a}^{\mathcal{W}}(t-\triangle t) \\ \mathbf{a}^{\mathcal{W}}(t) = \mathbf{a}^{\mathcal{W}}(t-\triangle t) + \triangle t\ \nu_a(t-\triangle t) \\ \mathbf{q}^{\mathcal{W}}_{\mathcal{C}}(t) = \exp(\triangle t\ \omega^{\mathcal{W}}(t-\triangle t))\mathbf{q}^{\mathcal{W}}_{\mathcal{C}}(t-\triangle t) \\ \omega^{\mathcal{W}}(t) = \omega^{\mathcal{W}}(t-\triangle t) + \triangle t\ \nu_\omega(t) \\ \mathbf{g}^{\mathcal{W}}(t) = \mathbf{g}^{\mathcal{W}}(t-\triangle t) \\ \lambda(t) = \lambda(t-\triangle t) \end{cases}$$

Here the noise terms $\nu_a(t)$, $\nu_\omega(t)$ represent respectively linear jerk and angular acceleration. The measurement model is given by the following equations:

$$\mathbf{y}_{\text{acc}}(t) = \begin{bmatrix} (\mathbf{q}^{\mathcal{W}}_{\mathcal{C}}(t))^{-1}(\mathbf{a}^{\mathcal{W}}(t) + \mathbf{g}^{\mathcal{W}}(t)) \\ \|\mathbf{g}^{\mathcal{W}}(t)\| \end{bmatrix} + \eta_{\text{acc}}(t)$$

$$\mathbf{y}_{\text{gyr}}(t) = (\mathbf{q}^{\mathcal{W}}_{\mathcal{C}}(t))^{-1}\omega^{\mathcal{W}}(t) + \eta_{\text{gyr}}(t)$$

$$\mathbf{y}_{\text{cam}}(t) = \begin{bmatrix} \mathbf{t}^{\mathcal{W}}(t) \\ \mathbf{q}^{\mathcal{W}}_{\mathcal{C}}(t) \end{bmatrix} + \eta_{\text{cam}}(t)$$

i.e., the acceleration measurements from the IMU comprise gravity acceleration, which is assumed of constant norm (hence the second componenent of $\mathbf{y}_{\text{acc}}$ is a fictitious measurement), the gyroscopes measure angular velocities and PTAM tracker provides, through $\mathbf{t}^{\mathcal{C}}_{\mathcal{W}}$, a measurement of the unscaled position $\mathbf{t}^{\mathcal{W}}$ and of the attitude.

A simplified version of EKF can be devised by excluding the estimation of the scale and exploiting the acceleration measurements only to the purpose of retrieving attitude information. In fact, in our context the gravitational acceleration $\mathbf{g}$ is much bigger then the acceleration due to the device's movements $\mathbf{a}_d$ and noise η, i.e.: $\mathbf{a}_{acc} = \mathbf{a}_d + \mathbf{g} + \eta$, with $|\mathbf{g}| \gg |\mathbf{a}_d| \sim |\eta|$, so separating device acceleration by subtracting gravity from the accelerometer readings is not easy: a small relative error in the estimation of gravitational acceleration leads to a big relative error in the estimation of device acceleration. For this reason the problem of acceleration estimation is ignored altogether: the device's acceleration is incorporated in the noise term and the accelerometer is assumed to measure only gravity. Furthermore, in order to reduce the non linearities in the measurement model, the dynamics is written in terms of the variables directly provided by the PTAM tracker. The resulting discrete-time model is:

$$\begin{cases} \mathbf{t}^{\mathcal{C}}_{\mathcal{W}}(t) = \mathbf{t}^{\mathcal{C}}_{\mathcal{W}}(t-\triangle t) + \triangle t\ \mathbf{v}^{\mathcal{C}}_{\mathcal{W}}(t-\triangle t) \\ \mathbf{v}^{\mathcal{C}}_{\mathcal{W}}(t) = \mathbf{v}^{\mathcal{C}}_{\mathcal{W}}(t-\triangle t) + \triangle t\ \nu_v(t-\triangle t) \\ \mathbf{q}^{\mathcal{C}}_{\mathcal{W}}(t) = \exp(\triangle t\ \omega^{\mathcal{C}}(t-\triangle t))\mathbf{q}^{\mathcal{C}}_{\mathcal{W}}(t-\triangle t) \\ \omega^{\mathcal{C}}(t) = \omega^{\mathcal{C}}(t-\triangle t) + \triangle t\ \nu_\omega(t) \end{cases}$$

The two noise terms $\nu_v(t)$ and $\nu_\omega(t)$ represent respectively a linear acceleration and an angular acceleration. The measurement model is given by the following equations:

$$\mathbf{y}_{\text{acc}}(t) = \mathbf{q}^{\mathcal{C}}_{\mathcal{W}}(t)\ \mathbf{g}^{\mathcal{W}} + \eta_{\text{acc}}(t)$$

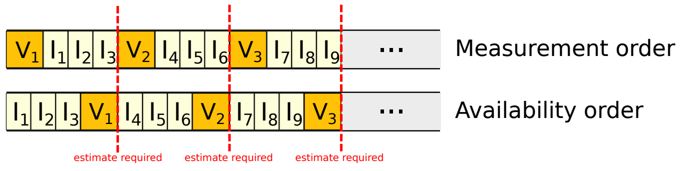

Fig. 5. Measurement order and availability order.

$$\mathbf{y}_{\text{gyr}}(t) = -\omega^{\mathcal{C}}(t) + \eta_{\text{gyr}}(t)$$

$$\mathbf{y}_{\text{cam}}(t) = \begin{bmatrix} \mathbf{t}^{\mathcal{C}}_{\mathcal{W}}(t) \\ \mathbf{q}^{\mathcal{C}}_{\mathcal{W}}(t) \end{bmatrix} + \eta_{\text{cam}}(t)$$

With the first version of the EKF that comprises all variables, including scale, the tracking problem could be completely solved, being able to estimate the pose in metric coordinates. Instead, we adopt the EKF simplified model described above for two main reasons. First, the filtering could improve the (unscaled) pose estimation in those situation where the PTAM tracking is temporarily lost and provides trajectory estimates characterized by abrupt deviations when in fact the trajectory is smooth. Second, the EKF could replace the Pose Prediction module (1, 2), by performing roughly the same function: estimating the preliminary camera pose for frame tracking. In our case this is motivated by the low power of a smartphone's cpu that makes it difficult to obtain good visual tracking accuracy while maintaining an high frame rate, and at the same time lowering the frame rate deteriorates the tracker's accuracy as it becomes increasingly difficult for the motion model to correctly guess the camera pose over greater time intervals. This is aimed at allowing the system to only track a reduced number of frames with good accuracy while still producing pose estimates at full frame rate. In this way sensor fusion becomes an integral part of the visual odometry algorithm, in contrast to the more common approach of fusing the two data sources a posteriori.

A. EKF implementation issues

Using the EKF in our system is not completely straightforward since the measurements are required to be ordered in time for the update step to be consistent, while in our system this is not the case. In fact during the time required by the tracking procedure to calculate the pose of a frame, several new sensor measurements are received from the IMU and used to update the EKF. Figure 5 schematize this problem: V_i indicates a visual measurement (a frame pose calculated by the tracking algorithm), and I_i indicates a sensor measurement. For example visual measurement V_2 is relative to a frame that precedes in time sensor measurements I_4, I_5 and I_6 but its tracking procedures is completed only after I_4, I_5 and I_6 have been received by the Sensor Fusion module. Moreover as explained in the previous Section, the exact arrival time of a sensor measurement is completely unpredictable, as it is unpredictable the time employed to track a frame.

Our sensor fusion algorithm is designed to address these problems using a sensor measurements queue. Its work-flow is detailed in Figure 4. Defining the EKF's state as the triple

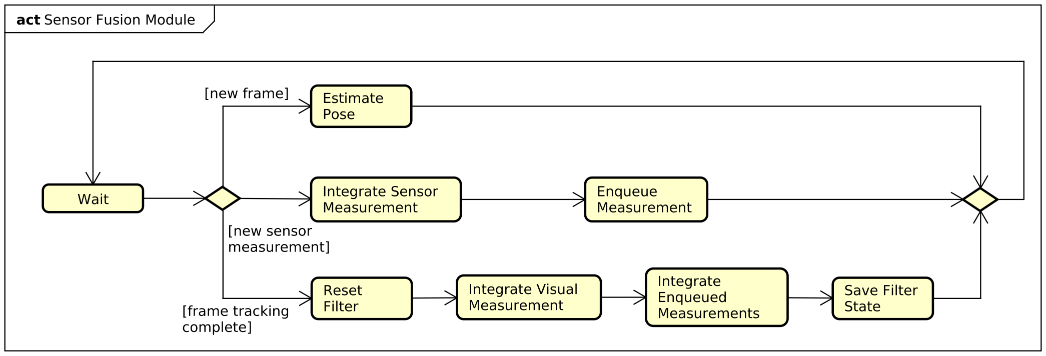

Fig. 4. Activity diagram of the Sensor Fusion module.

$(\hat{\mathbf{x}}(t|t), \mathbf{P}(t|t), t)$, i.e., the a-posteriori estimate, a posteriori covariance, and time, each time a new measurement is available, or an a priori pose estimate is requested by the tracking module, one of the following three actions takes place:

- When the tracking module requests an a priori pose, estimate it from the current state of the EKF.
- When a new sensor measurement is available, if its timestamp t_I is such that $t_I > t$ use it to update the EKF and append it to the sensor measurements queue.
- When a new visual measurement is available, reset the filter to its saved state and update the EKF with the visual measurement, then use all the sensor measurements in the queue to update the EKF and empty the queue. Finally save the new state of the filter.

This strategy ensures that measurements are always integrated in the correct order while using every available measurement to calculate the pose estimate.

We finally remark that the sampling interval $\triangle t$ is not constant, hence it is necessary to provide this information, that can be derived from the timestamps of the measurements, to the discrete-time dynamics of the EKF module.

V. Experimental Results

Here we report about the performance of the PTAM tracking module and of the EKF module correcting the pose estimation. Our experimental setup consisted of a rotating rod constraining the camera and sensor's movements along a circular trajectory with a radius of approximately 70cm. A number of test sequences has been captured moving the camera at varying angular speeds along a 90 degrees arc of this circumference.

A. Preliminary testing

A preliminary study has been conducted about the impact of the various building blocks of the original algorithm on the overall processing time. A direct porting of the original code has been ran on our Android testing device (a Google Nexus S) and the elaboration times analyzed with the help of a profiler. The results of this testing highlights the FAST corner detection step as a critical point. In fact, while other parts of the algorithm present a clear trade-off between tracking accuracy and speed of execution, in the form of execution times proportional to the number of map points used for tracking, the time spent for FAST corner detection is constant and cannot be easily optimized or reduced. At the same time it is quite easy to reduce computation times for the rest of the algorithm by reducing the number of map points tracked. Additionally, the results show that the number of map points that can be tracked in real-time by the testing device is very low, making the two-step pose refinement procedure superfluous.

B. Tracking accuracy and robustness

To test the validity of our adaption of PTAM's tracking we compared its output to that of the original PTAM software, using the original implementation of the mapping thread for both and applying them off-line to our test sequences. The Root Mean Square of the reprojection errors of map points is used as a metric of the similarity of the output of the two algorithms. If the original PTAM finds a certain map point j at position $\mathbf{p}_j^{\mathcal{C}_i}$ in the frame i, and our software finds the same point in the same frame at position $\hat{\mathbf{p}}_j^{\mathcal{C}_i}$ then the reprojection error RMS is defined as:

$$\text{RMS} = \sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{M_i} \|\mathbf{p}_j^{\mathcal{C}_i} - \hat{\mathbf{p}}_j^{\mathcal{C}_i}\|^2}{N \sum M_i}}$$

where N is the number of frames and M_i is the number of map points found in frame i. Figure 6 shows the values of this metric for the first 120 frames of two of our test sequences. For every sequence the tracking algorithm has been repeated varying two parameters: number of map points searched and frame rate ratio ρ. This last parameter is used to simulate the effects of long elaboration times, and is defined as: $\rho = r_{\text{cam}}/r_{\text{track}}$, where r_{cam} is the frame rate of the video source and r_{track} is the frame tracking rate of our software. A value of $\rho = 1$ corresponds to tracking at full frame rate, $\rho = 2$ means tracking every second frame and so on.

A first noteworthy observation regards the effects of changing the number of features: it can be seen that increasing this number doesn't always mean better accuracy. An explanation of this behavior can be found in the fact that the map points to be searched in every frame are selected giving priority to the most recognizable ones, so increasing the number of map

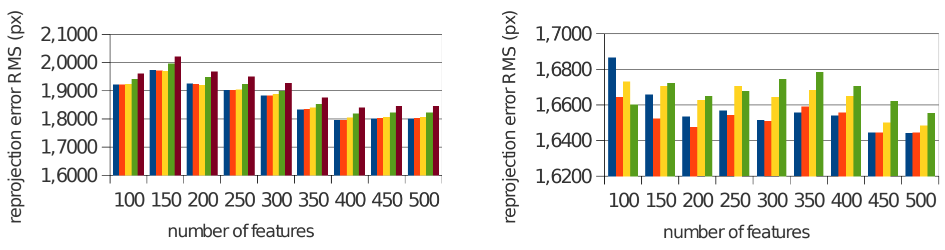

Fig. 6. Reprojection error RMS for two sequences. From blue to brown: $\rho = 1, 2, 3, 4, 5$

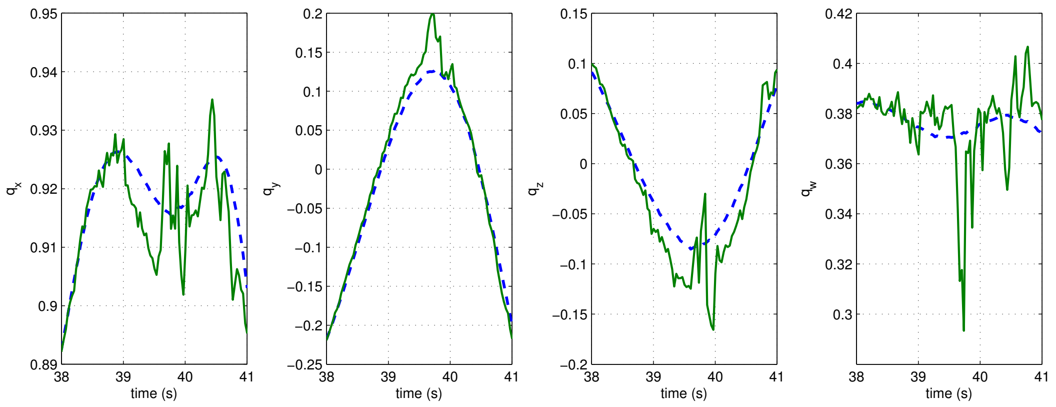

Fig. 7. Comparison between attitude estimation of the PTAM tracking (green, solid) and the EKF sensor fusion (blue dashed). The plots represent the x,y,z and w components of the orientation quaternion $\mathbf{q}^{\mathcal{C}}_{\mathcal{W}}$.

points means introducing progressively worse quality data into the system. In some cases this effect completely compensates the robustness added by increasing the number of features.

As expected, increasing values of ρ decrease tracking accuracy, as the larger distance between two consecutive camera poses renders the a-priori pose calculated by the Pose Prediction module more inaccurate. In general we can see a threshold-like behavior: tracking accuracy decreases gradually until a certain threshold value of ρ, over which the tracking fails altogether. This is evident in the second diagram of Figure 6, where the fifth data series is absent, meaning tracking failure.

C. *Sensor fusion performance*

Regarding the performance of the EKF sensor fusion, we compared the pose estimation provided by PTAM and by the EKF. The lack of a reliable ground truth for our test sequences makes only a qualitative evaluation of the results possible. We observed that the EKF usually provides small corrections with respect to the estimation given by the PTAM tracking. However, in several difficult to track sections of our test sequences, where PTAM's pose estimation is lost or severely inaccurate, the EKF is able to provide a more meaningful pose estimate than the visual algorithm alone. One such case is shown in Figure 8: the camera undergoes a fast but smooth direction change, resulting in blurry frames in a zone of the environment with few recognizable features. Figure 7 shows the attitude estimates provided by PTAM and by the EKF: it is clear that the estimate provided by PTAM alone is greatly inaccurate, while the EKF produces an estimate that is more compatible with the true camera motion.

For what concerns the use of the EKF to the purpose of substituting the Pose Prediction module, we report that at the current state of our implementation this did not result in a improvement of the performance for this module. Further work will be devoted to this issue in order to ascertain its potential utility.

VI. Conclusion

We presented a successful implementation on an Android device of an egomotion estimation algorithm by porting the tracking module of PTAM. This an important prerequisite toward a successful implementation of a mobile AR system for environmental monitoring. A main element of novelty of the proposed algorithm lies in a sensor fusion component, which has demonstrated to be effective on improving the position estimation and has potential to further be exploited for being integrated directly into the PTAM Pose Prediction module. Immediate future works include to extend the sensor fusion module in order to jointly estimate the scale and the pose and to integrate a mapping module into the current implementation.

Acknowledgment

The authors would like to thank Mauro Boccadoro for his support about Kalman Filtering.

This research has been partially funded by the European 7th Framework Program, under grant VENTURI (FP7-288238).

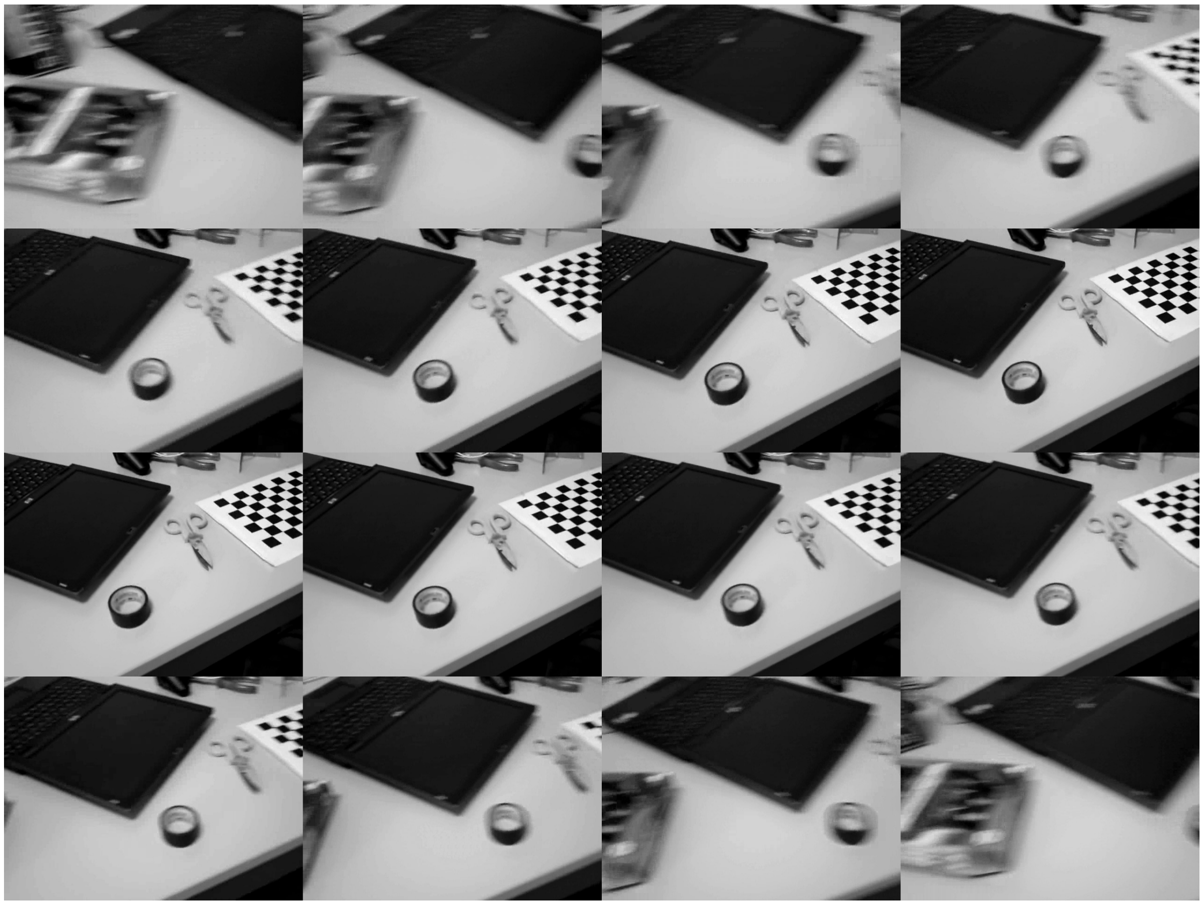

Fig. 8. Video frames sampled from the same section of the test sequence used in Figure 7. Only a frame every three of the original sequence is shown, resulting in a $\triangle t$ between the first and the last equal to $1.5s$.

REFERENCES

[1] M. Trincavelli, M. Reggente, S. Coradeschi, A. Loutfi, H. Ishida, and A. Lilienthal, "Towards environmental monitoring with mobile robots," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 2210 –2215, sept. 2008.

[2] E. Veas, R. Grasset, E. Kruijff, and D. Schmalstieg, "Extended overview techniques for outdoor augmented reality," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 18, pp. 565 –572, april 2012.

[3] M. Clothier and M. Bailey, "Augmented reality visualization tool for kings stormwater bridge," in *IASTED International Conference on Visualization, Imaging and Image Processing (VIIP)*, 2004.

[4] D. Goldsmith, F. Liarokapis, G. Malone, and J. Kemp, "Augmented reality environmental monitoring using wireless sensor networks," in *Proc. of the 12th Int. Conference on Information Visualisation*, 2008.

[5] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, 2007.

[6] *Eigen*. `http://eigen.tuxfamily.org/index.php?title=Main_Page`, 2012.

[7] Google INC. *Android SDK*. `http://developer.android.com/index.html`, 2012.

[8] T. K. Yaakov Bar-Shalom, X. Rong Li, *Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software*. Wiley-Interscience, 2002.

[9] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *In European Conference on Computer Vision*, 2006.

[10] J. B. Kuipers, *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality*. Princeton University, 2002.

[11] G. Klein and D. Murray, "Parallel tracking and mapping on a camera phone," in *Proc. Eigth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'09)*, 2009.

[12] "Special issue: 2nd workshop on integration of vision and inertial sensors," *International Journal of Robotic Research*, vol. 26, no. 6, pp. 515–517, 2007.

[13] F. Mirzaei and S. Roumeliotis, "A kalman filter-based algorithm for imu-camera calibration: Observability analysis and performance evaluation," *Robotics, IEEE Transactions on*, vol. 24, pp. 1143 –1156, oct. 2008.

[14] G. Nuetzi, S. Weiss, D. Scaramuzza, and R. Siegwart, "Fusion of imu and vision for absolute scale estimation in monocular slam," *Journal of Intelligent and Robotic Systems*, vol. 61, pp. 287–299, 2011.

[15] T. Oskiper, Z. Zhu, S. Samarasekera, and R. Kumar, "Visual odometry system using multiple stereo cameras and inertial measurement unit," in *In IEEE Conference on CVPR07*, 2007.

[16] E. Jones and S. Soatto, "Visual-inertial navigation, mapping and localization: A scalable real-time causal approach," *The International Journal of Robotics Research*, vol. 30, no. 4, pp. 407–430, 2011.

[17] J. Kelly and G. S. Sukhatme, "Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor self-calibration," *International Journal of Robotics Research*, vol. 30, no. 1, pp. 56–79, 2011.

[18] M. Mammarella, G. Campa, M. Napolitano, M.L. Fravolini, M. Perhinschi, Y. Gu, "Machine Vision / GPS Integration Using EKF for the UAV Aerial Refueling Problem", *IEEE Transactions on Systems, Man, and Cyber and Cybernetics, Part C: Applications and Reviews*, vol. 38, no. 6, pp 791-801, 2008.

[19] G. Campa, M. Napolitano, M.L. Fravolini, "A Simulation Environment for Machine Vision Based Aerial Refueling for UAV", *IEEE Transaction on Aerospace and Electronic Systems*, vol. 45, no. 1, pp. 138-151, 2009.

[20] E. Ricci and J.-M. Odobez. "Learning large margin likelihoods for realtime head pose tracking", *Int. Conf. Image Processing (ICIP)*, 2009.