6-DoF Pose Localization in 3D Point-Cloud Dense Maps Using a Monocular Camera

Carlos Jaramillo, Ivan Dryanovski, Roberto G. Valenti, and Jizhong Xiao*, *Senior Member, IEEE*

***Abstract*—We present a 6-degree-of-freedom (6-DoF) pose localization method for a monocular camera in a 3D point-cloud dense map prebuilt by depth sensors (e.g., RGB-D sensor, laser scanner, etc.). We employ fast and robust 2D feature detection on the real camera to be matched against features from a virtual view. The virtual view (color and depth images) is constructed by projecting the map's 3D points onto a plane using the previous localized pose of the real camera. 2D-to-3D point correspondences are obtained from the inherent relationship between the real camera's 2D features and their matches on the virtual depth image (projected 3D points). Thus, we can solve the Perspective-n-Point (*PnP*) problem in order to find the relative pose between the real and virtual cameras. With the help of *RANSAC*, the projection error is minimized even further. Finally, the real camera's pose is solved with respect to the map by a simple frame transformation. This procedure repeats for each time step (except for the initial case). Our results indicate that a monocular camera alone can be localized within the map in real-time (at QVGA-resolution). Our method differentiates from others in that no chain of poses is needed or kept. Our localization is not susceptible to drift because the history of motion (odometry) is mostly independent over each *PnP + RANSAC* solution, which throws away past errors. In fact, the previous known pose only acts as a region of interest to associate 2D features on the real image with 3D points in the map. The applications of our proposed method are various, and perhaps it is a solution that has not been attempted before.**

I. INTRODUCTION

Depth sensors continue to gain popularity as technology progresses and algorithms for indoor 3D mapping (also known as scene modeling) become more robust. Modeling of indoor environments with an RGB-D (red, green, and blue color image, plus depth information) sensor like the Microsoft Kinect has advanced rapidly due to its affordability, availability, and accuracy for depth acquisition indoors. RGB-D mapping continues to improve as shown by the state-of-the-art techniques in [1] and [2]. However, the uniqueness of the presented work is the use of a monocular camera (an ubiquitous sensor) to do localization within a point cloud map produced by any type of depth sensors such as lasers, stereo cameras, time-of-flight cameras, RGB-D cameras, etc. 3D mapping still remains a difficult problem, and our approach to localization in a point-cloud map using a simpler sensor (i.e., monocular camera) has rarely been investigated before.

After a 3D map of a static scene is available, we prove that it is possible to re-enter the same scene with a camera alone and estimate its 6-degree-of-freedom (6-DoF) pose: 3D translation and rotation relative to the map. There are plenty of applications of our proposed method: augmented reality showcases and games, museum tours, inspection, mobile robot navigation, etc., can all benefit from localization within a 3D map. The ubiquity and unconstrained motion of monocular cameras, such as in smartphones and mounted in aerial vehicles, requires that the localization gets processed on-board, which our method qualifies for. Another robotics application is for swarm navigation, where a leader equipped with powerful sensor(s) can map the environment first, so its followers carrying simple cameras can localize themselves in the map.

As far as we know, we are the first at attempting to localize a monocular camera in an environment modelled as a 3D point cloud. Note that we are not attempting to solve the Visual Simultaneous Localization and Mapping problem using a single camera (MonoSLAM [3]). Various real-time methods for relocalization using a monocular camera have been proposed. For example, in 2007 by Williams, Klein and Reid [4], and similar biologically-inspired systems given in [5], [6]. With the emergence of cost-effective RGB-D sensors, Engelhard, Endres, and Hess [7] developed a 3D Visual SLAM. Faster and more accurate visual mapping techniques continue to improve, as in [2] by Dryanovoski, Valenti and Xiao.

Monocular SLAM methods based on PTAM (Parallel Tracking and Mapping), such as [4], include a tracking thread with a similar functionality to our proposed method. Although [4] uses monocular images (without relying on a 3D map), their approach is different because our relocalization method does not need to keep a history of features in the map, but only the previous 3D pose of the camera is needed as an aid to look around the area in which the camera was previously at. Older information is discarded on a frame-to-frame basis, as opposed to the PTAM solution, which tracks and bundle adjusts by growing the map (some form of SLAM). Our advantage of having a dense 3D map is enormous (computationally speaking and for robustness), but mainly because the camera's egomotion is not a factor for localization within the map (assuming the previous known pose is more-or-less correct).

This work is supported in part by U.S. Army Research Office under grant No. W911NF-09-1-0565, U.S. National Science Foundation under grants No. IIS- 0644127 and No. CBET-1160046, Federal HighWay Administration (FHWA) under grant No. DTFH61-12-H-00002 and PSC-CUNY under grant No. 65789-00-43.

Carlos Jaramillo is with the Dept. of Computer Science, The Graduate Center, The City University of New York (CUNY), 365 Fifth Avenue, New York, NY 10016 (e-mail: cjaramillo@gc.cuny.edu)

Ivan Dryanovski is with the Dept. of Computer Science, The Graduate Center, The City University of New York (CUNY), 365 Fifth Avenue, New York, NY 10016 (e-mail: idryanovski@gc.cuny.edu)

Roberto. G. Valenti is with the Electrical Engineering Department, City College of New York, Convent Ave & 140th Street, New York, NY 10031 (e-mail: rvalent00@ccny.cuny.edu)

*Jizhong Xiao is with the Electrical Engineering Department, City College of New York, Convent Ave & 140th Street, New York, NY 10031 (*corresponding author, e-mail: jxiao@ccny.cuny.edu)

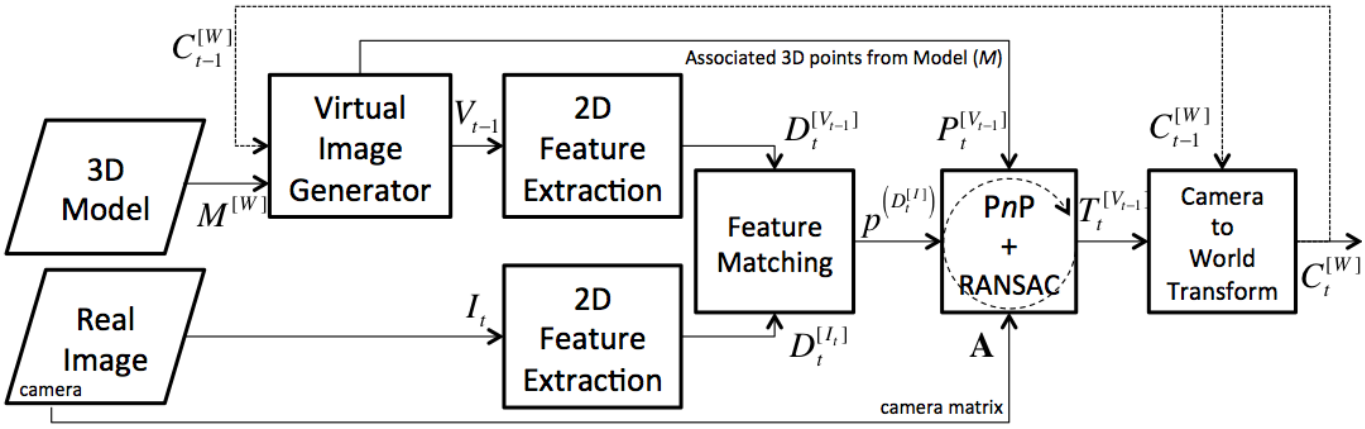


Fig. 1: Pipeline for the monocular localization at a given time t, where $t > 1$. The 3D map, $M^{[W]}$ is a static input, whereas the real images, I arrive continuously. We generate a virtual image, V_{t-1} from the map and the previously known pose of the camera in the map, $C_{t-1}^{[W]}$. Then, 2D features, D_t are extracted and matched from both images. The matched 2D feature points, $p^{\left(D_t^{[I_t]}\right)}$, of the real image are refined against the associated 3D points, $P_t^{[V_{t-1}]}$, via *RANSAC* applied to *PnP*. This produces the relative transformation, $T_t^{[V_{t-1}]}$, between the real and the virtual camera frames, $[I_t]$ and $[V_{t-1}]$, respectively. The new camera pose in the global frame, $C_t^{[W]}$, is found after a simple transformation and it is final output of the pipeline at time t (and the only parameter that is saved for the next iteration, if any)

We rather follow the suggestion given in the Further Work section of [8], so the user initially maps out the scene in order to avoid resource-intensive SLAM techniques for re-entry localization. In fact, we don't track points as the camera moves in space, and there is where our contribution mainly outstands. Also, we use more precise and denser point-cloud maps (with color information). We believe any view inside the map serves to robustly localize our camera without drift due to egomotion tracking. Our solution is explained and demonstrated in the following sections.

II. Monocular Localization within a 3D Map

A. Overview of the Proposed Method

Since we do not try to solve the Visual SLAM problem, we assume the 3D point-cloud map, $M^{[W]}$, already exists and does not change (it is static) as the camera revisits it. (Note that the 3D points of the map are positioned with respect to the global frame $[W]$). At a given time step $t > 1$, the only dynamic input to our system is an image, I_t, from the real camera. A virtual image, V_{t-1}, is generated in parallel (at time t) from the previous pose, $C_{t-1}^{[W]}$, computed at time $t-1$ by projecting the 3D point cloud $M^{[W]}$ onto a virtual image plane (refer to Subsection II-D). Also, notice that we treat the initial case at $t = 1$ differently (see Subsection II-C).

Fast feature extraction and matching is performed with both images I_t and V_{t-1}, resulting in 2D feature points (keypoints) sets $D_t^{[V_{t-1}]}$ and $D_t^{[I_t]}$, respectively. Out of the matched virtual keypoints, $p^{\left(D_t^{[V_{t-1}]}\right)}$, from $D_t^{[V_{t-1}]}$, we know their constructing 3D points, $P_t^{[V_{t-1}]}$ with respect to the virtual camera frame, as well as their 2D keypoint correspondences, $p^{\left(D_t^{[I]}\right)}$, on the "real" image. We use these 3D points to solve the perspective from n points problem (*PnP* [9]) with respect to the virtual camera frame, $[V_{t-1}]$. Consequentially, we apply *RANSAC* [10] to reject outlier matches from the original feature sets based on their reprojection error. Thus, from *PnP* and *RANSAC*, we obtain the relative transformation, $T_t^{[V_{t-1}]}$, between the image frames $[V_{t-1}]$ and $[I_t]$ at the current time t. Finally, we compute the new pose, $C_t^{[W]}$, of the camera with respect to the map (global frame coordinates $[W]$) using the previous known position, $C_{t-1}^{[W]}$, of the camera, which is now transformed by the relative pose, $T_t^{[V_{t-1}]}$:

$$C_t^{[W]} = C_{t-1}^{[W]} \cdot T_t^{[V_{t-1}]} \tag{1}$$

Thus far, we have briefly described a time-step of the process, which repeats indefinitely. A high-level overview of our method is given in the pipeline of Fig. 1. In the next subsections, we explain the essential components of this pipeline. However, we begin by briefly describing how the 3D point-cloud maps are generated offline for use with our localization approach.

B. Offline 3D Point-Cloud Mapping

Our method requires an existing 3D map, M, as a set of 3D points with color information, which is constructed *offline* by some methods, such as with iterative-closest point (ICP [11]), as done in [2], [12]. In other words, ICP is used to align the individual, consecutive 3D point-clouds (a.k.a. keyframes), which are also saved in order to compute the initial camera pose as described in Subsection II-C). Newcombe et al. [13] presented a system for RGB-D pose tracking and mapping by aligning dense depth data against a map of a surface. However, their system requires a GPU-equipped computer. Another map builder based on feature descriptors is given by Endres et al.

Fig. 2: A dense 3-D point cloud map that was built real-time (for offline use with our method) using an RGB-D sensor at 640x480 pixels resolution at a rate of 30 frames per second. This map was constructed as given in [2].

[14], but it also requires a powerful processor, so it is not an attractive solution for low-performance on-board robots.

Instead, Dryanovski, Valenti and Xiao proposed a real-time visual odometry and mapping system for RGB-D cameras [2]. We use this method to build dense, point-cloud maps of static, indoor environments. Then, our method can localize the pose of any monocular camera that revisits the mapped scene. An example of a dense map built with this system is shown in Fig. 2.

C. Initial Pose Estimation

The pipeline of Fig. 1 can only be used when a previous pose, $C_{t-1}^{[W]}$, is available. However, at $t = 1$, this is not true, so we need another way of computing $C_0^{[W]}$. In a nutshell, we estimate the initial pose by comparing the first "real" image, I_1, against every keyframe image used to produce the 3D point-cloud map. We take advantage of knowing the map is a data structure of keyframes, K_j, where $j = \{1, \ldots, J\}$ for a map composed of J keyframes and their poses. Each keyframe is composed of a pair of color image and its registered depth image, or $(I_{RGB}, I_{depth})_{K_j}$. More details about the map data structure can be found in [12] and [2].

In the first input image, I_1, we detect SURF features [15] and its descriptors, $D_1^{I_1}$, and we do the same for all the keyframes' color images, $I_{RGB,K_j}, \forall K_j$. Next, we train a descriptor matcher from all the features across the J keyframes. For each feature descriptor in the real image, we find n nearest feature neighbors using the descriptor matcher. Each feature in I_1 will, therefore, point to a vector of descriptor matches, which contain relevant information such as keypoint index and matching distances. For all n vectors of descriptor matches, we sort them based on the matching distance (in feature descriptor space), and we take the top n candidates (the first descriptor match of each vector).

Finally, we perform a robust *PnP* matching between the n points from the real image and their corresponding 3D points in the map obtained from the top n matches. Note that feature keypoints in the descriptor matches can resolve to their parent keyframe index, j, so 3D points in the global frame $[W]$ are immediately given by its registered depth image, I_{depth,K_j} and its global pose in the map (also available in the data structure). Additionally, we employ *RANSAC* on the *PnP* in order to refine the initial pose $C_0^{[W]}$ that maximizes the number of inlier keypoints.

The initial pose estimation is a slow procedure, which depends on the CPU characteristics, the size of the map, and the feature matching parameters such as the n number of neighbors sought across the J map keyframes, the feature descriptor type, as well as the *RANSAC*-related parameters (e.g. number of iterations, threshold distance, etc.).

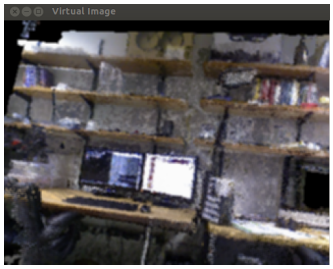
D. Virtual Images from Perspective Projection

Procedure GenerateVirtualImage($M^{[W]}$, A, w, h, $C_k^{[W]}$)

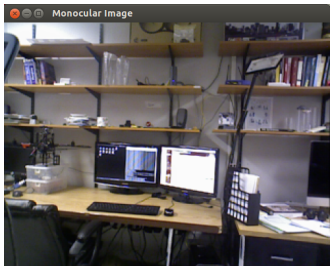
Input: $M^{[W]}$: Dense 3D point-cloud map (frame $[W]$)
Input: A: the 3×3 intrinsic camera matrix
Input: w: the camera's width (in pixels)
Input: h: the camera's height (in pixels)
Input: $C_k^{[W]}$: The known pose (rotation matrix Rot and translation vector $\mathbf{t}$) for the camera frame I_k with respect to the global frame $[W]$ at time k.
Output: V_k: Color virtual image at time k
Output: R_k: Depth (range) virtual 2D image at time k

```
// Initialize color and depth buffers
V_k[All] ← 0
R_k[All] ← 0
foreach i, where i = {1, ..., ||M||} do
    p^[W] ← M^[W].3Dpoints[i]
    // Transform world point P into the camera frame
    p^[V_k] = Rot * p^[W] + t
    d ← p^[V_k].z  // Depth

    if d > 0 then
        // Perform projection into the virtual image plane V_k
        v_k = A · p^[V_k]
        x ← v_k.x; y ← v_k.y; z ← v_k.z
        // Determine image pixel (u, v)
        u ← x/z; v ← y/z
        // Accept point only within the camera's field-of-view
        if (u < w) and (u ≥ 0) and (v < h) and (v ≥ 0) then
            // Compute depth buffer
            if R_k[u, v] == 0 then
                V_k[u, v] ← p^[W].color ; R_k[u, v] ← d
            else
                // Smaller depth is saved
                if (d > 0) and (depth < R_k[u, v]) then
                    V_k[u, v] ← p^[W].color ; R_k[u, v] ← d
return V_k, R_k
```



(a) Virtual image, V_{t-1}



(b) Real image, I_t

Fig. 3: Image examples: **(a)** Virtual image V_{t-1} generated by the projection of the 3D point cloud (3D map) at the last known pose $C_{t-1}^{[W]}$ **(b)** Real image I_t from the monocular camera at time t

Fig. 4: Resulting SURF feature correspondences between real (left) and virtual (right) images (shown in Fig. 3b and Fig. 3a, respectively).

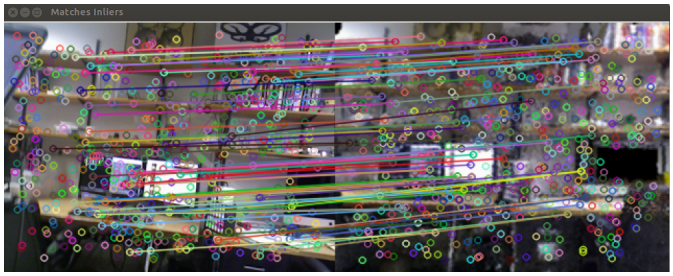


Fig. 5: Filtered feature correspondences of matches shown in Fig. 4 after executing *PnP* with *RANSAC* (reprojection error used as fitness = 10 pixels) .

An essential component of our system is the generation of the virtual image V_k at time k, where $k = t-1$. In addition, we compute a virtual depth image, R_k, that encodes the associated 3D position of the points $p^{[V_k]}$ in the map as seen by the virtual camera. The `GenerateVirtualImage` procedure explains our algorithm thoroughly. As input, the procedure requires the map, $M^{[W]}$, and the intrinsic matrix for the pinhole camera model,

$$A = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2}$$

where f_x and f_y are the focal lengths, and (c_x, c_y) is the optical center of the image. The remaining input arguments are the width and height, (w, h), of the desired virtual images (measured in pixels), and the last known pose $C_k^{[W]} = [\ Rot \quad \mathbf{t}\ ]$.

We zero-initialize the image and depth buffers (V_k, R_k). Then, we iterate over the $\|M\|$ 3D points of the map, $M^{[W]}$. Each point $\mathbf{p}^{[W]}$ is transformed into the virtual camera frame, $[V_k]$, as follows:

$$\mathbf{p}^{[V_k]} = Rot * \mathbf{p}^{[W]} + \mathbf{t} \tag{3}$$

The point $\mathbf{p}^{[V_k]}$ is then transformed onto the virtual image frame as $\mathbf{v}_k = A \cdot \mathbf{p}^{[V_k]}$. The pixel coordinates (u, v) in the image is obtained by simply dividing the x and y components of $\mathbf{v}_k$ by its z component, respectively.

We project only those points within the camera's field-of-view. We select the nearest points by maintaining a Z-buffer, which is eventually used as the resulting virtual depth image R_k (needed for reconstructing the 3D points in some region of interest at time $t = k+1$). Since the point cloud is not solid (dense enough) and its projection from a different viewpoint can produce holes in the virtual image, we fill them up by interpolating to the value of each point nearest neighbor pixel. Fig. 3a is an example of a virtual image V_{t-1} generated at some time t using the last known pose $C_{t-1}^{[W]}$.

E. Feature Matching

The second stage of the pipeline (Fig. 1) is to use the virtual image, V_{t-1}, and the real image, I_t, (examples are Fig. 3a and Fig. 3b, respectively) to find feature correspondences. In order to avoid extracting point features in regions of the virtual image where depth values are missing (color was interpolated), we construct a "mask" from the virtual depth image, R_{t-1}, and pass it to the feature detector function for the virtual image.

We have experimented with several choices of feature detectors, including SURF, ORB, and Shi-Tomasi keypoints. While our implementation offers a configurable choice between various feature detectors, we arbitrarily chose the SURF detector due to its robustness for handling rotation and scale variance and localization efficiency [16]. Note that features are detected on the intensity channel of the images. Feature correspondences are found through the Fast Library for Approximate Nearest Neighbors (FLANN) [17]. An example of some found feature correspondences is shown in Fig. 4.

F. Relative Pose Transformation with PnP *and* RANSAC

In order to compute $C_t^{[W]}$, as given by (1), we need to estimate the relative pose of the camera, $T_t^{[V_{t-1}]}$, at time t. For this purpose, we perform *PnP* on the 2D feature points of the real image, $p^{(D_t^{[I]})}$, with their 3D point correspondences, $P_t^{[V_{t-1}]}$, which have been encoded in the virtual depth image, R_{t-1}, and are given with respect to the virtual camera frame, $[V_{t-1}]$.

We also run *RANSAC* to refine the relative pose estimates $T_t^{[V_{t-1}]}$ that *PnP* computes. Inliers are selected from feature matches (correspondences) according to some threshold reprojection error (in pixels). Fig. 5 presents an example of this final refinement.

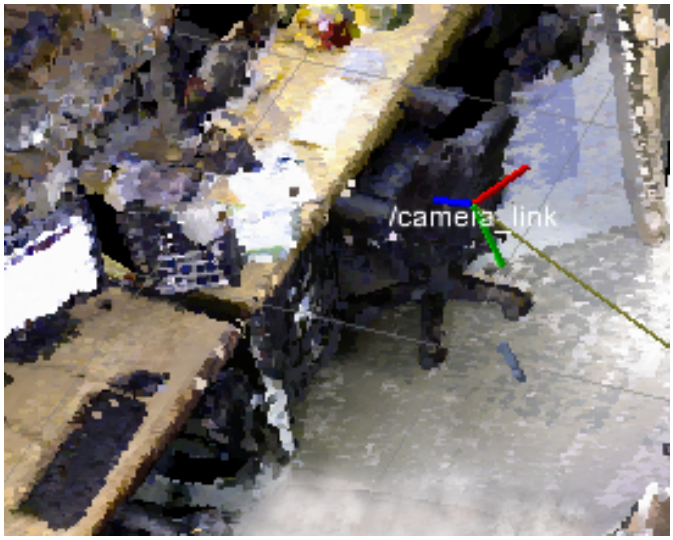


Fig. 6: Visualization of an estimated pose, $C_t^{[W]}$, of the monocular camera at some time t. Visualized axis in red (X), green (Y), and blue (Z) corresponds to the camera frame where Z is the optical axis pointing forward, and X, Y is the image plane

Once the current $C_t^{[W]}$ pose is estimated with respect to the global frame, we repeat the pipeline, by updating $C_{t-1}^{[W]} \leftarrow C_t^{[W]}$ with the latest known pose, and we throw away any previous poses. Recall that our localization method does not depend on the camera's odometry (chain of pose transformations).

III. Experimental Results

A. System Implementation

1) We build the dense 3D map, $M^{[W]}$, as a point cloud using an RGB-D sensor (e.g. Microsoft Kinect) by applying mapping techniques such as the proposed by [2] and shown in Fig. 2.
2) We implement the system described in Section II, on top of open-source tools such as the OpenCV library [18] and the Point Cloud Library (PCL) [19], both available in the Robotics Operating System (ROS) framework [20]. The implementation is in continuous development, and is available from our `ccny-ros-pkg` repository.

Fig. 4 is a snapshot of the successful localization (at some arbitrary time interval t) inside the dense 3D map is shown in Figure 7 for 320x240 pixels images and SURF feature detection with approximately 60 inliers.

B. Experiments

Our current results are mostly qualitatively due to the lack of "dense enough" ground truth datasets obtained from RGB-D sensors. A simple error analysis is performed in Subsection III-C. A supporting video (http://youtu.be/0O28HHFl4VU) reflects the The supporting video reflects the/ appropriate pose estimation of the camera within the 3D map. For a camera at QVGA resolution (320x240 pixels), the average execution times running on a 1.7 GHz Intel Core i5 processor (inside a virtual machine) were:

- Virtual Image Generation: 70 ms
- SURF feature detection and description: 100 ms
- SURF Feature matching with FLANN: 8 ms
- *PnP* with *RANSAC* (1000 iterations, 50 inliers, 10 pixels reprojection error): 200 ms

Bear in mind that these time values include the visualization overhead of the 3D map and the images.

C. Error analysis

We have only compared pairwise error using the pose of the RGB camera in the sensor, as a ground truth reference for the error metric of the system and its relocalization in the same 3D map with the proposed method.

This error metric is based on [21]. It operates among the transformation estimated by the proposed system, $\hat{C}_t^{[W]}$, and the transformation reported for the corresponding keyframe pose, $C_t^{[W]}$, at time t. This error is formally defined as

$$E = \sum_{i=1}^{n} \left(\hat{C}_t^{[W]} \ominus C_t^{[W]} \right) \tag{4}$$

where the $\ominus$ operator on two transforms A and B is

$$A \ominus B \equiv B^{-1} A \tag{5}$$

The error E is a transform which includes a rotational matrix component E_{Rot}, and a translation vector component $E_{\mathbf{t}}$. We define the total pose estimation error, $e_{\mathbf{t}}$, as the size of the translation error vector:

$$e_{\mathbf{t}} = \| E_{\mathbf{t}} \| \tag{6}$$

We define the total angular error, e_{Rot}, as the principal angle of the rotation matrix error, E_{Rot}, given by

$$e_{Rot} = |\cos^{-1}(0.5 \ \mathrm{tr}(E_{Rot}) - 1)| \tag{7}$$

From our experiment, the average error is about 4 cm (translational) and 1 degree (rotational). However, we are aware that this metric is not sufficient to determine the precision of the system under all situations since here, the camera views are also keyframe images that created the original map. A more quantitative error analysis should be performed by tracking the monocular camera with a motion-capture system.

IV. Conclusion

We presented a pose localization method for a monocular camera in a in a prebuilt 3D point-cloud map of a static environment. We computed matches between the real and virtual images by using fast and robust 2D features . A virtual view (color and depth images) was constructed by projecting the map's 3D points onto a plane using the localized pose of the real camera from the previous time step (except for the first pose). 2D-to-3D point correspondences were obtained from the inherent relationship between the real camera's 2D features and their matches on the virtual depth image (constructed from

3D points in the map). Then, we solved the Perspective-n-Point (*PnP*) problem to find out the relative, 6-DoF transformation between the real and virtual cameras. Further refinement is done with *RANSAC*, so the projection error is minimized and inlier correspondences result for the computed relative pose transformation. From a final frame transformation, the result is the real camera's 6-DoF pose (3D position and rotation) with respect to the map (global frame). This procedure repeats indefinitely for each time step.

Our results indicate that with an appropriate choice of parameters, a monocular camera can be localized in the 3D map in real-time (for QVGA-resolution images on a laptop's single-core CPU). Our method differentiates from others in that no chain of poses is necessary, but only the last known pose is preserved to act as a region of interest in order to associate 2D features on the real image with 3D points in the map. As a result, the localization of the camera is independent from odometry since *PnP + RANSAC* relocalize the camera at each time step. In contrast, a chain of transformations tend to accumulate error (drift) as what happens with visual odometry solutions. We believe there are many applications that fit into this scenario and our camera localization method in 3D point-cloud maps provide a solution to this unique and challenging problem that has not been adequately investigated before.

V. Discussion and Future Work

Although our preliminary results look very promising because we are able to run the algorithm in real time on a modern computer, there are some caveats we need to address. We are aware that the quality of the produced virtual images affects the feature correspondence procedure. Indeed, lots of weak features affect the performance of the entire algorithm, and it is important to mention that the robustness of the correspondences is an important aspect to consider. The fine-tuning of *RANSAC* parameters, such as the maximum number of iterations, the pixel reprojection error, and the number of inliers, are essential for speeding up the *PnP* computation with robust results. As already mentioned, we compute the initial pose of the camera by means of stored keyframes from the mapping stage. In fact, this first step adds an initial delay before the live image-feed can start to be processed. We also need to validate our method by experimenting with bigger maps and performing error analysis with ground truth data.

We propose several enhancements for the application of this method such as aiding the rotation estimation with IMU sensors (smart phones usually have them readily available) and using wider field-of-view real (as well as virtual) in order to tolerate drastic motion. The generation of virtual images can be speed up by referring to smaller regions of interest, which can only be obtained from spatially organized point-cloud data structures such as Octomaps or KD-trees. Also, we are aware that working with dynamic environments can be difficult (for now maps must be static). Perhaps, masking foreground objects from both map and visiting images can help workaround this problem.

References

[1] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D Mapping: Using depth cameras for dense 3D modeling of indoor environments," in *the 12th International Symposium on Experimental Robotics (ISER)*, 2010.

[2] I. Dryanovski, R. G. Valenti, and J. Xiao, "Fast Visual Odometry and Mapping from RGB-D Data," in *International Conference on Robotics and Automation*, vol. 10031, 2013.

[3] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: real-time single camera SLAM." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, Jun. 2007.

[4] B. Williams, G. Klein, and I. Reid, "Real-Time SLAM Relocalisation," *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, 2007.

[5] M. J. Milford and G. F. Wyeth, "Single camera vision-only SLAM on a suburban road network," *2008 IEEE International Conference on Robotics and Automation*, pp. 3684–3689, May 2008.

[6] M. J. M. Milford and G. G. F. Wyeth, "Mapping a Suburb With a Single Camera Using a Biologically Inspired SLAM System," *Robotics, IEEE Transactions on*, vol. 24, no. 5, pp. 1038–1053, 2008.

[7] N. Engelhard, F. Endres, and J. Hess, "Real-time 3D visual SLAM with a hand-held RGB-D camera," in *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, Vasteras, Sweden, 2011.

[8] G. Klein and D. Murray, "Parallel Tracking and Mapping on a camera phone," *2009 8th IEEE International Symposium on Mixed and Augmented Reality*, pp. 83–86, Oct. 2009.

[9] F. Moreno-Noguer, V. Lepetit, and P. Fua, "Accurate Non-Iterative O(n) Solution to the PnP Problem," *2007 IEEE 11th International Conference on Computer Vision*, pp. 1–8, 2007.

[10] M. Fischler and R. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, 1981.

[11] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp," *Proc. of Robotics: Science and Systems ...*, 2009.

[12] I. Dryanovski, C. Jaramillo, and J. Xiao, "Incremental registration of RGB-D images," *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[13] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pp. 127–136, Oct. 2011.

[14] F. Endres, J. Hess, D. Cremers, and N. Engelhard, "An Evaluation of the RGB-D SLAM System," in *International Conference on Robotics and Automation*, vol. 1, no. c. Ieee, May 2012, pp. 1691–1696.

[15] H. Bay, A. Ess, T. Tuytelaars, and L. Vangool, "Speeded-Up Robust Features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, Jun. 2008.

[16] T. Tuytelaars and K. Mikolajczyk, "Local Invariant Feature Detectors-A Survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1391082

[17] M. Muja and D. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Application VISSAPP'09)*, vol. 340. INSTICC Press, 2009, pp. 331–340.

[18] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, 2008.

[19] R. B. Rusu, S. Cousins, and W. Garage, "3D is here: Point Cloud Library (PCL)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 1–4.

[20] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *International Conference on Robotics and Automation (ICRA)*, no. Figure 1, 2009.

[21] J. Sturm, S. Magnenat, F. Colas, D. Cremers, N. Engelhard, R. Siegwart, F. Pomerleau, and B. Wolfram, "Towards a benchmark for RGB-D SLAM evaluation," *RSS 2011 Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, pp. 1–2, 2011.