Visual Autonomy via 2D Matching in Rendered 3D Models

D. Tenorio, V. Rivera, J. Medina, A. Leondar, M. Gaumer, and Z. Dodds(✉)

Harvey Mudd College CS Department, Claremont, CA 91711, USA
zdodds@gmail.com

Abstract. As they decrease in price and increase in fidelity, visually-textured 3D models offer a foundation for robotic spatial reasoning that can support a huge variety of platforms and tasks. This work investigates the capabilities, strengths, and drawbacks of a new sensor, the *Matterport 3D camera*, in the context of several robot applications. By using hierarchical 2D matching into a database of images rendered from a visually-textured 3D model, this work demonstrates that – when similar cameras are used – 2D matching into visually-textured 3D maps yields excellent performance on both global-localization and local-servoing tasks. When the 2D-matching spans very different camera transforms, however, we show that performance drops significantly. To handle this situation, we propose and prototype a map-alignment phase, in which several visual representations of the same spatial environment overlap: one to support the image-matching needed for visual localization, and the other carrying a global coordinate system needed for task accomplishment, e.g., point-to-point positioning.

1 Motivation and Context

This project investigated the strengths and drawbacks of a sensor that is relatively new for robot systems: a *Matterport camera* [1]. As shown in Fig. 1, the camera offers a new level of ease and fidelity to the creation of visually-textured 3D maps. Although made more ubiquitous by the Kinect [2], the use of such rich, visual 3D representations is still relatively new as a foundation for robot spatial reasoning. Considerable work has shown the power of 3D-to-3D matching, e.g., [3–6] and many others, but the underlying 3D visual maps are far less well studied for robot platforms with our era's most accessible and least expensive sensor source: a run-of-the-mill RGB camera. Bridging this 2D-3D gap will allow mobile robots to move autonomously within rich environmental models using inexpensive sensors.

This work developed an image-matching framework to allow robots of several types – with only 2D image input – to recognize their position with respect to a 3D (Matterport) model. Ideally, any robot with a camera would be able to utilize this work's algorithms to determine its position within a visually-textured 3D model; in practice, we found that different imaging transforms can cause significant degradation of matching accuracy. To compensate, we prototyped a system in which local 2D visual maps from different camera sources are aligned to the 3D model, providing both accurate 2D visual matching and corresponding 3D coordinates within the model's global coordinate system.

G. Bebis et al. (Eds.): ISVC 2015, Part I, LNCS 9474, pp. 373–385, 2015.
DOI: 10.1007/978-3-319-27857-5_34

Thus, this work demonstrates both the challenges and the promise emerging from Matterport models and other rich, visually-textured 3D maps. Specifically, we contribute

- the use of Matterport's sensor to create 3D maps suitable for a variety of robot tasks
- a three-tier approach for matching a robot's live-acquired 2D images with a database of known-location renderings from a 3D visual Matterport model
- a quantitative assessment of both the accuracy and speed of that matching process
- task-applications implemented on several different robot platforms, providing insights into the matching-process's strengths (accuracy and speed sufficient to support real-time control) and drawbacks (less robustness across distinct cameras/ image sources) – along with an alignment step that mitigates the drawbacks

Overall, there are compelling reasons for considering Matterport models as a basis for robot control: they match our intuition of our shared environs as a "human visual world" (even though we subconsciously construct it, moment-to-moment), they are the space in which humans plan and execute tasks, and thus they are the space in which humans most naturally specify tasks for *robot* planning and execution. This work helps introduce and inform both roboticists and vision researchers of what Matterport's models can (and can't) contribute to their efforts.

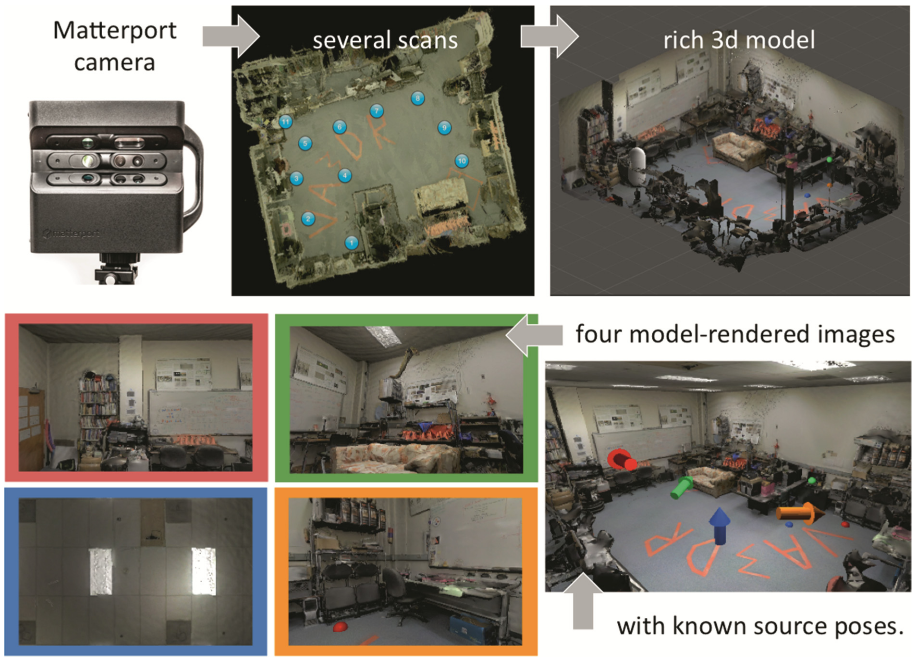


Fig. 1. Matterport camera and a lab-environment model, along with several rendered images from that model and the *known* locations from which those images were rendered.

2 Processing the Matterport Models

Computationally, the fundamental problem we tackle is that of comparing a (live) 2D image with a 3D model created both at an earlier time and from perspectives different than the current image. Our approach to this problem is split into two parts: a preliminary processing of the 3D environmental model(s) and robust 2D image matching.

We made a comprehensive model of several environments beforehand. Figure 1 shows screenshots from one that served as our primary robot workspace; others from our lab and elsewhere are available from [1]. To create these models, we first took multiple scans of the room with the Matterport camera, at about 45 s per scan. Following the company's standard process, the data was sent to Matterport, where it was compiled into a photorealistic 3D model of the environment. The default interface, however, allows navigation only from and to the locations at which scans were taken in the real world, noted in the top row's middle panel. This constraint prevents us from being able to create renderings – with accompanying coordinates – except from those environmental viewpoints.

This problem was solved by downloading the full environmental mesh from and processing it with Unity 3D [7]. Its rendering engine allows arbitrary navigation of the space, rendering a 2D view of the model from any location inside it or out, and even from poses far from source-scan locations, e.g., Figure 1's top-right rendering.

2.1 From 3D Model to 2D Renderings

We leveraged this geometric freedom to create a database of 2D images of the environment. Inside the three-dimensional model, we programmed a virtual camera to move to each lattice point within a large grid, taking one screenshot facing each wall at every position, creating a set of 200 2D images that covered the space: many of those renderings had significant field-of-view overlap. We built several such databases for testing; in the final version, we rotated the camera 90° between each rendering in order to minimize the number of pictures in the database. Certainly, Unity 3D makes it straightforward to create sparser or denser sets of renderings if the additional resolution is needed for a specific task. Crucially, the Unity-based system recorded and remembered the pose at which each rendering was made: thus, each image "knew" where it had been taken. Figure 1 (bottom row) shows four of these rendered images, along with an overview showing the location and direction of the virtual camera's location for each.

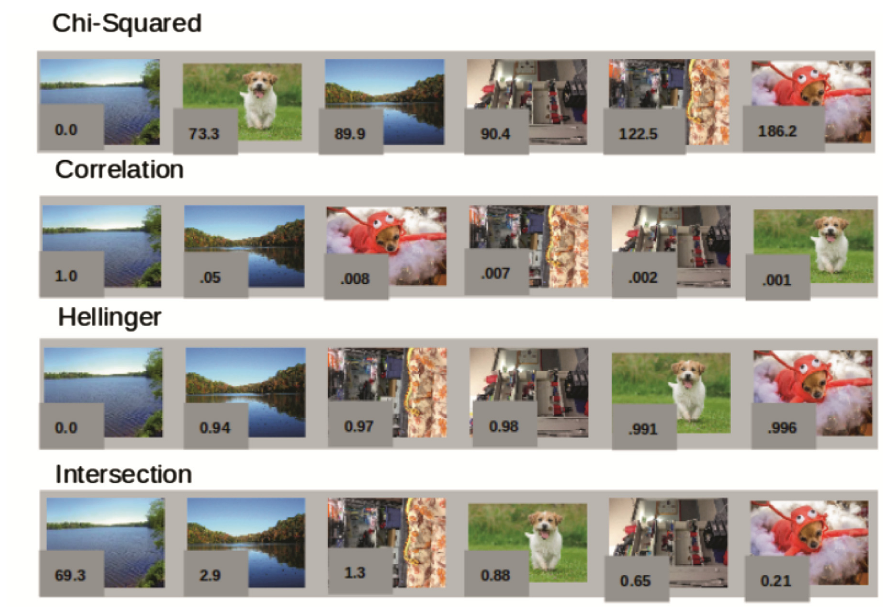
3 Matching into the Database of Rendered Images

To facilitate finding the best match between a robot's current image and the images in the database we combined several image comparison algorithms using OpenCV 3.0's Python bindings. The resulting system used three steps in order to match a novel (live) image with the database of renderings described in Sect. 2:

1. First, color-histogram matching is performed in order to narrow the search.
2. Geometric matching with ORB features further narrows the set of possible matches

3. Finally, several geometric constraints are considered: the database image that best satisfies those constraints is used as the best match for the task's current timestep

Where the database's resolution suffices, robot tasks use the location of the best-matched image as an approximation of the current robot location: Sect. 4 shows that for some aerial navigation tasks, this suffices. Further pose refinement is possible, however, from the image information in the corresponding ORB features; Sect. 4 also highlights other tasks, e.g., robot-homing that leverage this additional precision. This section details the three-step matching process and presents its results.



	Bad (%)	Par. Good (%)	Good (%)	Number of Images Used
Color Algs. - Score	13.33	13.33	73.33	15
ORB Algs. - Score	93.33	7.67	0	15
ORB Algs. - Vote	7.67	7.67	86.67	15
All Algs. - Vote	4.5	37	58.5	200
Homography Check	0	0	100	200

Fig. 2. (top) Examples of the scores produced by the four color-histogram-matching algorithms used in the image-matcher's first pass; (bottom) the overall match results both for individual tiers of the image-matching algorithm, top four rows, and the last-row final result. With layers of color-based filtering, keypoint-based matching, and geometric consistency checks, *all* novel rendered images matched very closely to the 200-image database.

3.1 Pixel-Based and Color-Histogram Matching

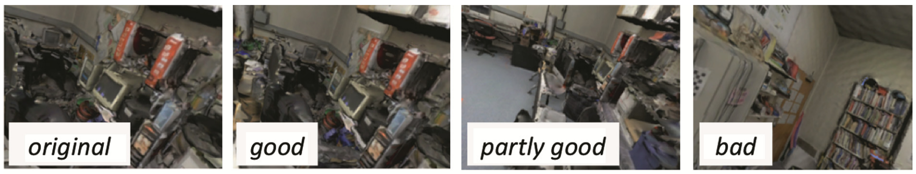
The first group of algorithms used in the image matching process consisted of seven scoring thods using color-histogram and pixel-by-pixel comparisons. The three pixel-by-pixel comparisons included are mean-squared error (MSE), root MSE, and the structural similarity index [8]. We also use four color-histogram comparison algorithms adapted from [9]. For each image, we discretized its data into N = 16 colors (histogram bins). Four different histogram-comparison methods provided by OpenCV [10], named *Correlation*, *Chi-Square*, *Intersection*, and *Hellinger*, were then used in order to measure the similarity of a novel image with each database image. Figure 2 (top) shows the resulting scores of a one-vs-all comparison using a small seven-image database obtained both from [9] and our environment's images. Here, the test image (a lake scene) is included in the comparison in order to show how a perfect match score compares to the imperfect match scores. In all subsequent tests, however, including those summarized in the results in Fig. 2 (bottom), the test image was *not* included in the comparison set.

As Fig. 2's lower table shows, even a small test database of 15 images from our lab environment showed only mediocre accuracy, when based on color-based matching alone (top row). This prompted the addition of a second tier of algorithms, which added geometric information to the matching process via small-patch image features.

3.2 Feature-Based Matching

The second group of algorithms relied on an image's geometric information. A menagerie of feature types exist with SIFT [11] and SURF [12] two of the most commonly used. Following [13], ORB keypoints, i.e., Oriented FAST and Rotated BRIEF keypoints, detect and compute distinctive image locations at a speed more than ten times greater than that of SURF (and SIFT). That speed guided our choice of ORB. Using OpenCV's ORB implementation, our image-matcher considers five geometric scores: (a) the sum of all ORB-matched visual distances, (b) the median of those distances, (c) their mean, (d) the sum of the visual distances of the top 50 % of matches by visual similarity, and (e) the number of inlier matches based on geometric (homography) consistency, as described in Sect. 3.3.

Preliminary tests involving the accuracy of the image matching system (Fig. 3, at right) showed that, like the color matching, the geometric matching alone did not provide accuracy suitable for supporting robot navigation. Having obtained a database of 200 images from the 3D model of the lab environment, a subset of fifteen images was run against the other images in the set and found the best match as determined by the system. The result was then assessed by eye, noting each as *good*, *partially good*, or *bad*. The (human assessment) metric the team used for these three categories, along with examples of each, appear in Fig. 3, below. Again, no image was included within the database set against which it was matched.



Grade	*Location in Room of Match Relative to Current Image*
Good	Same location in room; very little movement needed, if any.
Partially good	Similar location; some translation or rotation would be required to be at the same location as the current image.
Bad	Entirely different location in room.

Fig. 3. Examples and criteria for judging the categories *good*, partly or *partially good*, and *bad* image matches: Good image matches are one of the nearest neighbors in a sampled direction; partly good matches overlap less closely; bad matches do not overlap.

In addition to comparing color-only and feature-only matching, we compared two different methods for determining the "best" match: a "voting" system and a "scoring" system. A total of 12 match-scores are computed (7 pixel- and color-based and 5 ORB-based, as noted above): those 12 are our feature vector for how well two images match. The "voting" system gives a single point, or vote, to the best-matching image for each of these 12 algorithms, with the winner having the largest number of votes. By contrast, the normalized "scoring" approach first normalizes each algorithm on a commensurate scale of 0.0 to 1.0 and then sums the 12 individual scores. The highest overall score is named the best match.

For each tier image-matching layer by itself, the "voting" system yielded more accurate results – this is due to the fact that a single bad score can sink an otherwise excellent match due to the color or feature differences in the small portion of the two images that do *not* overlap. However, even the voting system produced a completely incorrect match for 20 % or more of the images examined. To improve this result, we post-processed the best matches to determine the amount of visual overlap they contained.

3.3 Improving Match Quality via Geometric Constraints

Upon examining the results carefully, the source of all of the poor and many middling matches was an accidentally well-matched set of image keypoints across two frames with similar color compositions. Figure 4 shows a typical example of a poorly-aligned and a well-aligned set of keypoints from example matches. As long as there are four matched keypoints – and every possible match included more than four – the transformation of those keypoints' locations from the test image to the database image yields a 2D-2D homography that transforms the first image plane into the second. When the keypoints are inconsistently matched, the result is a dramatic – and obviously incorrect – warping of the original scene, as Fig. 4's top example illustrates.

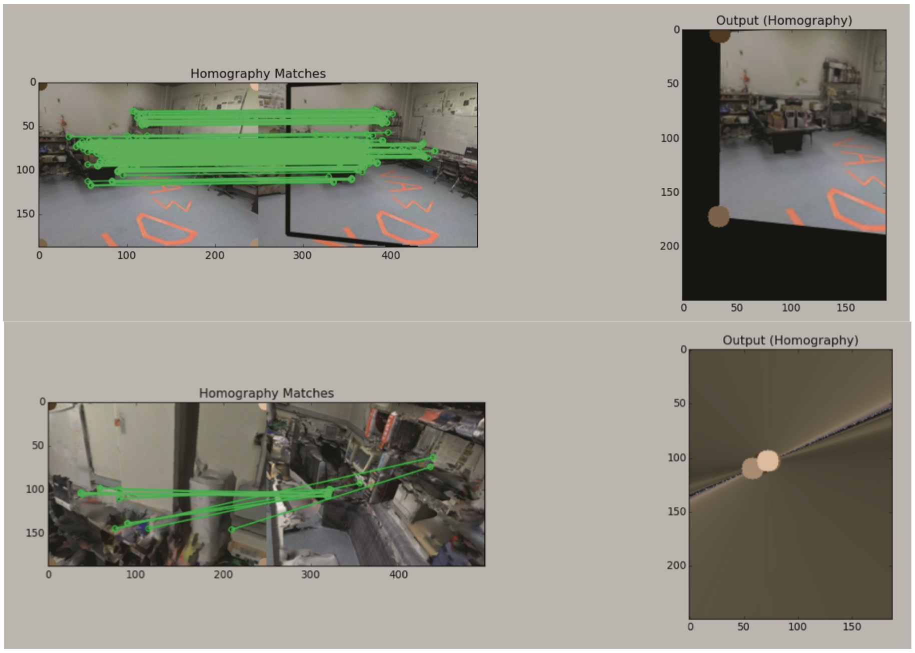


Fig. 4. (top) A poorly-aligned group of keypoints, yielding to a homography-transformed original that has "collapsed" into a very small image, a sign that the match is incorrect; (bottom) a well-aligned group of keypoints, with significant overlap: a strong match.

As such, we could determine when a bad match had been made by examining the Euclidean distance of the four corners post-homography. If the transformed corners pull too close to (or too far from) each other, the transformed image has experienced a "collapse" (or its reverse), and the system eliminates it from contention, continuing on in its examination of other candidate matches from the database.

With this homography-based geometric-consistency check, the results were encouraging. With the same database of 200 images as in the preliminary tests, now in a leave-one-out analysis, the test of every one of the 200 images produced an accurate, or *good* match. Thus, this homography check eliminated the bad matches without affecting the good ones, which ware those with significant image overlap, e.g., Figure 4's second example.

Though his paper's approach does benefit from not requiring camera calibration, these geometric constraints are *not* the only ones that might benefit this application. For instance, the P3P algorithm [17], among other 2D/3D relationships such as the epipolar constraint [16, 18], leverage the 3D model more deeply in order to support image-feature matching.

3.4 Results: From Accuracy to Speed

Encouraged by the accuracy of these results, we also considered the time required to match a novel image with a set of K database images. For use in robotic applications, speed translates directly into task-capability and -responsiveness. In addition, time-per-database-image determines how large, either in spatial extent or pose resolution, the environment's rendered database can be, while still supporting the ultimate goal of the project: autonomous, visual navigation. Thus, we examined how different sets of algorithms performed in terms of time per image, with the results shown in Fig. 5:

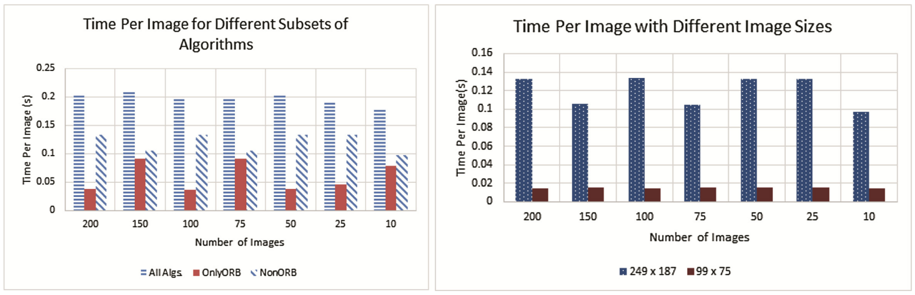


Fig. 5. (left) The time used in second to execute the full matching system on full-size (249 × 187) images, as well as its two component layers: ORB indicates the geometric components; NonORB, the color-based components of the algorithm; (right) the speedup obtained when decimated (99 × 75) images are used for the color-based matching.

Noticing the smaller time per image for ORB algorithms when compared with non-ORB algorithms for the default image size of 249 × 187 (Fig. 5, left), we explored whether smaller images could significantly improve the efficiency of the color-based matching. Although the ORB algorithms did not perform well at a smaller image size, the color algorithms did. The change to 99 × 75 pixel images – only for the color-based matching – improved the run-time by approximately an entire order of magnitude (Fig. 5, right). The significantly smaller time-per-image solidified the design decision to use color-based matching as a first pass over the database. The final image-matching system, as well as both of these timing tables, employ precomputed histograms and precomputed ORB features for the database images. For the test (novel) image, however, the time to compute those features is included.

4 Validation via Visual Navigation Tasks

With accurate and fast results, we deployed and evaluated the image-matching system in service of several autonomous, visual-navigation robot tasks:

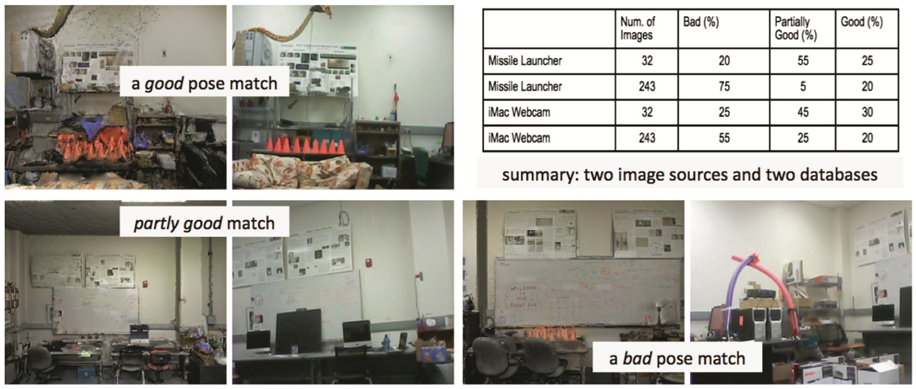
1. Image-based localization using a pan/tilt rig and a webcam: two live-image cameras *distinct* from the one used to image the database
2. Image-based homing on a differential-drive wheeled vehicle, comparing odometric homing accuracy with the matching system's image-based feedback
3. Point-to-point navigation of an aerial vehicle whose only sensor is a forward-facing camera

Each of these tasks – and its results – is detailed in the following subsections. For these tasks, the system used a 3D visually-textured model, in .obj format, via the widely used Unity3d rendering engine. Though obtained by a Matterport camera, the approaches detailed here apply to any such model in a compatible format.

4.1 Localization Using Different Source Cameras

We sought to test the functionality of the 2D image-matching under imaging conditions very different than the source, the Matterport sensor. To so this, input images were drawn from a pan/tilt rig with an inexpensive webcamera (as a side note, this platform is also a Nerf-based missile launcher) attached and the higher-quality camera built-in to a desktop iMac. Using images from either the launcher's built-in camera or the computer's webcam, the system compares those images to the database of 2D images from the model using our image matching program, and indicates the location at which the image was taken in the 3D model. Building from ROS's infrastructure [14], a client/server system connected the novel images (taken and compared on the server side) with the rendering of the resulting estimated camera position (by an arrow, as done in Fig. 1).

As Fig. 6 attests, the results were far less accurate than those obtained using the same live- and database-source cameras. Here again, three examples are provided: one each of *bad*, *partially good*, and *good*, this time equivalence classes for the ***pose*** of the result. In each case the best-matched database image is on the left, and the novel image is on the right (these, taken from the iMac's built-in camera):



	Num. of Images	Bad (%)	Partially Good (%)	Good (%)
Missile Launcher	32	20	55	25
Missile Launcher	243	75	5	20
iMac Webcam	32	25	45	30
iMac Webcam	243	55	25	20

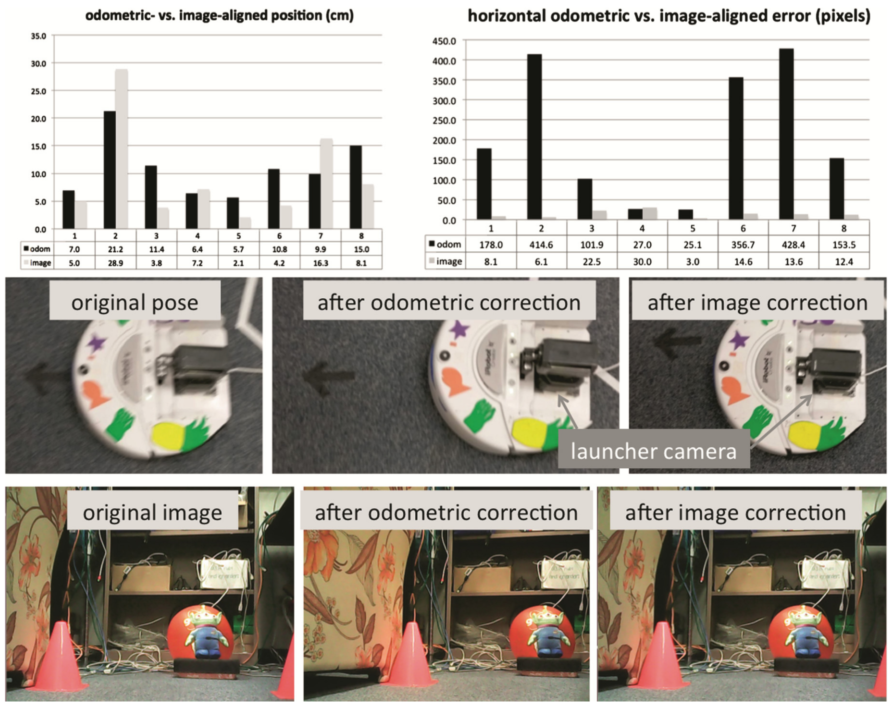
Fig. 6. Example pose matches and their results when using two different source cameras, each distinct from the original Matterport model's. Both a small (32-image) and a larger (243-image) database of environmental 2D renderings were compared. The strong accuracy results, above, are ***not*** replicated when the novel-image sources differ from the model's.

Combined with the excellent accuracy of the final matching system presented in Sect. 3 and Fig. 3, these results suggest several things. First, with different source cameras, the system is less accurate for larger databases: this results from the significantly different color response of each camera's sensor. The larger underlying database seems to provide more opportunities for distractors, causing a lower accuracy there than for the smaller set of source images. Most surprising to us was that, when the iMac's camera and even the low-quality launcher's webcam were used as *both* the source of both novel *and* stored images, the system performed at 100 % (or near-100 %) accuracy: it was not the absolute quality of the imaging sensor that mattered but its *consistency*.

4.2 Visual Homing Versus Odometric Homing on a Wheeled Robot

To leverage that fact for the two subsequent tasks (wheeled and aerial navigation), we overlaid the 3d environment's rendered images in a task-dependent way with images from the wheeled robot's camera, which was another launcher camera riding atop an iRobot Create, and with images from the drone's camera – the built-in sensor in a Parrot AR.Drone quadcopter. The wheeled vehicle was then run through eight *homing* tasks: from a given initial position – and initial image – the robot was driven randomly for several seconds in order to move far from that starting location.

The goal from there is to return home, i.e., to return to the initial position through a combination of odometry and image matching. First, the wheel-rotation sensors, which are quite noisy on the differential-drive Create robot [15], bring the robot back to the location believed to be the odometric starting point –not entirely accurate due to wheel



odometric- vs. image-aligned position (cm)

	1	2	3	4	5	6	7	8
■ odom	7.0	21.2	11.4	6.4	5.7	10.8	9.9	15.0
■ image	5.0	28.9	3.8	7.2	2.1	4.2	16.3	8.1

horizontal odometric vs. image-aligned error (pixels)

	1	2	3	4	5	6	7	8
■ odom	178.0	414.6	101.9	27.0	25.1	356.7	428.4	153.5
■ image	8.1	6.1	22.5	30.0	3.0	14.6	13.6	12.4

Fig. 7. Overall pose-accuracy results and an example of a wandered and corrected trajectory (top row), along with an example of a bird's-eye view of the odometry-vs-image-matching alignment (middle row) and the alignment of the images themselves (bottom row)

slippage. From there, the image-matching program is used to further refine the robot's position, by minimizing two components of the image-error: the average scale-change, Δs, among the ORB features and their average horizontal image-translation, Δx, to align the current image as close as possible to the initial image's view. Here, the goal is investigating the use of the images, not the control strategy: the robot alternates making small Δs and Δx corrections until it is no longer improving. Figure 7 summarizes all 8 and illustrates one homing run.

4.3 A Point-to-Point Task on a Vision-Only Quadcopter

A third spatial-reasoning task used to test the image-matching system combined the insight from the first localization trials (the significantly improved performance of having similar source and live cameras) with the coordinate system provided by the original image-matching task (using the 3D model of the lab environment). To do this,

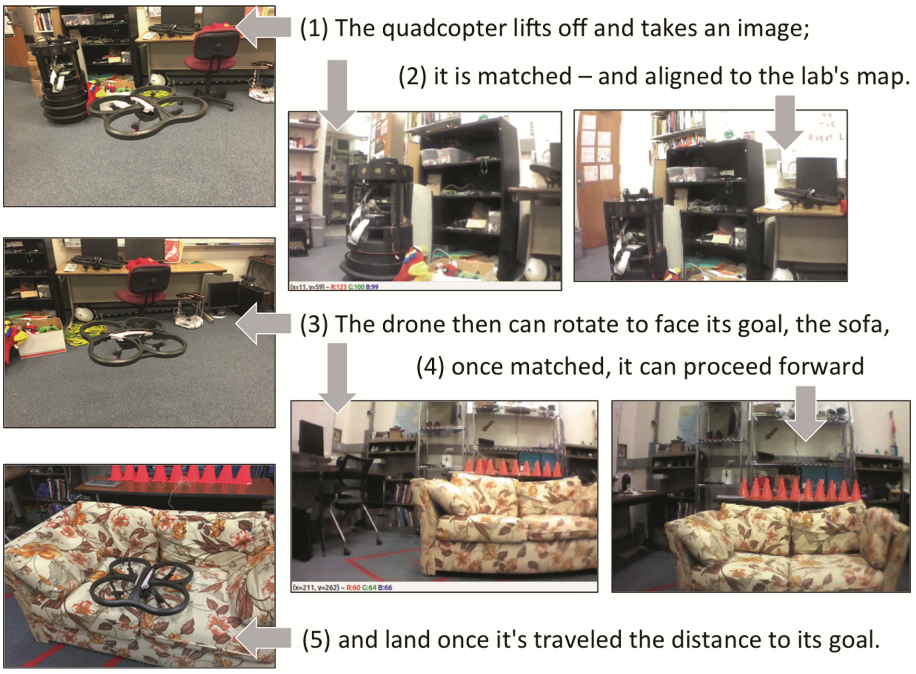

(1) The quadcopter lifts off and takes an image;
(2) it is matched – and aligned to the lab's map.
(3) The drone then can rotate to face its goal, the sofa,
(4) once matched, it can proceed forward
(5) and land once it's traveled the distance to its goal.

Fig. 8. Shows the use of aligned image-matching to accomplish a point-to-point aerial task. (**first image**, at upper left) Here, a quadrotor helicopter ascends and hovers in an unknown location within its environment. (**second image**, upper right) The image from the quadcopter (or "drone") appears at left and is matched with the image rendered from the lab map at right. That matching process (detailed in Sect. 3) results in a known, well-localized pose for the drone. From there (**third image**, middle left), the drone rotates in place until it is facing its desired destination (the lab couch). The process repeats (**fourth image**, lower right) with the drone-taken image matching a map-rendered image to trigger a state-change to proceed forward. Finally (**fifth image**, lower left), the drone lands at its desired pose.

an overlay of images from the forward-camera of a Parrot AR.Drone quadcopter was placed atop the coordinate system of the 3D model. Although the quadcopter's source images did not carry any information about their location in the lab environment, the 3D model to which those images were aligned did. In addition, because the images were regularly spaced, only one image needed to be hand-aligned in each drone location; the position of the rest could be inferred from the first. Although one could imagine a fully automatic alignment from the drone camera to the environment-model's camera, that process was not done for this set of tests. Figure 8 summarizes the results of a point-to-point task in which the drone sought out the lab environment's sofa, used image-matching to determine its orientation in the room, rotated to face the goal, again matched to confirm its rotational pose and determine its distance, and subsequently made the final motion required to land on the couch.

5 Verdict

Inspired by the capability and potential of the Matterport camera's 3D visual models as a basis for robot spatial-reasoning tasks, this work sought to (1) develop a 2D image-matching system to allow (robots with) ordinary cameras to localize within a Matterport model (2) assess the performance of that image-matching, and (3) validate the system through three different robot tasks. The three-tier image-matching system developed here demonstrated and validated both the accuracy and speed required to support autonomous vision-based robot tasks, but *only when the model and live source cameras were the same*. Here, we accommodated that limitation by aligning images from the robot itself to those portions of the 3D model necessary for task success, e.g., to a single location in the case of robot homing and with several rotational scans of the lab for the point-to-point navigation task with the aerial robot.

Although heartened by these successes, we believe this work highlights an important, perhaps underappreciated, challenge in using 3D models such as Matterport's for robot applications. As such models become more common, more opportunities will arise to query them with distinct sources of image information. Automatic *photometric* calibration between a Matterport camera and a (different) robot camera thus seems a tantalizing – and worthwhile – challenge. When combined with a robust 2D image-matching system, such as investigated in this work, that combination would tap even more of the robotics and spatial-reasoning potential of those richly-textured 3D maps.

Acknowledgments. We acknowledge and thank the generous support of the NSF, though CISE REU Site project #1359170, as well as from Harvey Mudd College and its Computer Science department.

References

1. Matterport. matterport.com. Accessed 20 October 2015
2. Oliver, A., Kang, S., Wunsche, B.C., MacDonald, B.: Using the Kinect as a navigation sensor for mobile robotics. In: Proceedings of IVCNZ 2012, pp 509–514, Dunedin, New Zealand. ACM, NY
3. Olesk, A., Wang, J.: Geometric and error analysis for 3D map-matching. In: Proceedings of IGNSS Symposium, 14 p, Queensland, Australia (2009)
4. Pinto, M., Moreira, A.P., Matos, A., Sobreira, H., Santos, F.: Fast 3D map matching localisation algorithm. J. Autom. Control Eng. **1**(2), 110–114 (2013)
5. Endres, F., Hess, J., Engelhard, N., Sturm, J.: In: Proceedings of ICRA 2012, pp. 1691–1696, 14–18 May 2012
6. Wu, C., Clipp, B., Li, X., Frahm, J.-M.: 3D model matching with viewpoint-invariant patches (VIP). In: Proceedings of CVPR 2008, pp. 1–8, 23–28 June 2008
7. Unity. unity3d.com. Accessed 20 August 2015
8. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Process. **13**(4), 600–612 (2004)
9. Rosebrock, A.L Pyimagesearch. www.pyimagesearch.com/
10. The OpenCV Library [http://opencv.org/]; Bradski, G. The OpenCV Library Dr. Dobbs Journal, November 2000
11. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. **60**(2), 91–110 (2004)
12. Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: SURF: speeded up robust features. Comput. Vis. Image Underst. (CVIU) **110**(3), 346–359 (2008)
13. Rublee, E., et al.: ORB: an efficient alternative to SIFT or SURF. In: 2011 IEEE International Conference on Computer Vision (ICCV). IEEE (2011)
14. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.Y.: ROS: an open-source robot operating system. In: Proceedings of Open-Source Software Workshop of the International Conference on Robotics and Automation (ICRA) (2009)
15. Watson, O., Touretzky, D.S.: Navigating with the Tekkotsu Pilot. In: Proceedings of FLAIRS-24, pp. 591–596, Palm Beach, FL, USA, 18–20 May 2011
16. Hartley, R., Zisserman, A.: Multiple View Geometry. Cambridge University Press, Cambridge (2004)
17. Haralick, R., Lee, C., Ottenberg, K., Nolle, M.: Review and analysis of solutions of the three point perspective pose estimation problem. Int. J. Comput. Vis. **13**(3), 331–356 (1994)
18. Lynen, S., Sattler, T., Bosse, M., Hesch, J., Pollefeys, M., Siegwart, R.: Get Out of My Lab: Large-scale, Real-Time Visual-Inertial Localization. Science and Systems, Rovotics (2011)