An Image Wherever You Look!
Making Vision Just Another Sensor for AI/Robotics Projects

Andy Zhang, John Lee, Ciante Jones, Zachary Dodds
Harvey Mudd College 301 Platt Blvd. Claremont, CA 91711 USA
axzhang, johlee, cjjones, dodds@g.hmc.edu

Abstract

Visual sensing can be difficult to incorporate into undergraduate robotics and AI assignments. Images, after all, do not provide a direct estimate of the geometric conditions within the field of view. Yet vision is increasingly compelling as a part of undergraduate AI and robotics, given the centrality of pixels in our students' interactions with technology and each other. This paper shares a small-footprint framework designed to make visual sensing as easy to incorporate into AI projects and assignments, e.g., as a source of evidence for localization algorithms, as range sensors. The framework leverages (hand-built) circular panoramas and the image-matching capabilities provided by OpenCV's python library. An example localization project highlights its pedagogical accessibility and ease of deployment atop low-cost hardware and alongside other sensors.

The Challenge

A fundamental challenge in developing undergraduate AI/Robotics assignments is deciding *what to include*. So many engineering, computing, and algorithmic priorities are shared by AI/Robotics that the limited timeframe and high expectations for insight-per-experience payoff often precludes the integration of image processing with traditional AI/Robotics tasks such as localization and mapping.

Yet vision-based localization and navigation systems have a history that reaches to the earliest days of AI/Robotics (Nilsson 1969; Moravec 1983). Contemporary vision-based robots, leverage both the mature understanding of visual geometry, e.g., (Diel, et al 2005), and the power of data-driven visual lookup, e.g., (Johns et al. 2016). Moreover, vision and robotics together now support many other investigations, e.g., husbanding endangered marine species (Wilby, et al. 2016), tracking lions in their natural habitat (Ackerman 2013), even surveilling other robots (Moeys et al 2016). To help undergraduates tap into these trends, we created and tested a software framework -- for building and localizing within circular panoramas using vision. The software is designed for both accessibility and "tinkerability" (Resnick and Rosenbaum 2013), including (a) near-zero cost of materials, (b) the adaptability of the vision-based algorithms, including their spatial and angular resolution, and (c) the ease with which new users can deploy and modify the system.

Thus, this project contributes a small-footprint software scaffold that makes it easy for instructors creating projects, or students seeking to extend their sensor suite's reach, to use vision to support localization in AI/Robotics projects:

- the system uses only Python and OpenCV (Bradski 2013) to support a cooperatively-built graph of panoramas, with student-chosen angular and translational resolution, as the map from image data to location
- several image-matching algorithms and resolutions were considered; we have chosen a default combination that produces a reasonable tradeoff in accuracy and time-performance
- a student deployment of the framework shows its flexibility in contributing to other low-cost, easily-deployed sensors and hardware, in service to an indoor navigation task, *the kitchen run*.

Developed in the summer of 2016, this framework has not yet been tested in a classroom setting. Even as the system itself matures and evolves, however, its goal – making computer vision an accessible facet of undergraduate AI/Robotics – will continue to motivate and challenge the AI education community.

Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

The Framework

Components and Design

At its heart, the framework is an implementation of circular panoramas, entirely in Python, and using a standard OpenCV installation. The panoramas themselves are created by students with the software and are a good starting point in getting comfortable with OpenCV. Choosing a number of images per location, the students take that many pictures using their robot, before driving (or moving by hand) to another location. Automatically creating a graph of panorama locations was not a goal of the current system (though the idea could certainly motivate an ambitious student team!) What is more, the panoramas are *not* stitched together: they are simply an angle-indexed list of images attached to a known location. Figure 1 shows panoramas (as image sequences) images from two example locations in our laboratory space, along with an overhead view of that space and an example image-matching result.

These simple and flexible representations offer several advantages: This representation has several advantages, especially in terms of accessibility, e.g.,

- The graph can be task-specific, e.g., Figure 1 shows a tape ***x*** at each location along a wide hallway and a narrower path leading to a neighboring kitchen
- The angular resolution of the images is both easy to vary and easy to interpret
- Each pose has a unique source image to which it corresponds, making it easy to reason about the localization system's behavior.
- Any of OpenCV's image-matching algorithms can be used to determine the likelihood that a novel image matches one of the panorama's images. Crucially, a *likelihood* is determined (based on the number of matching features) – this value can then be incorporated into whatever algorithmic framework the assignment, project, or exercise seeks to support.

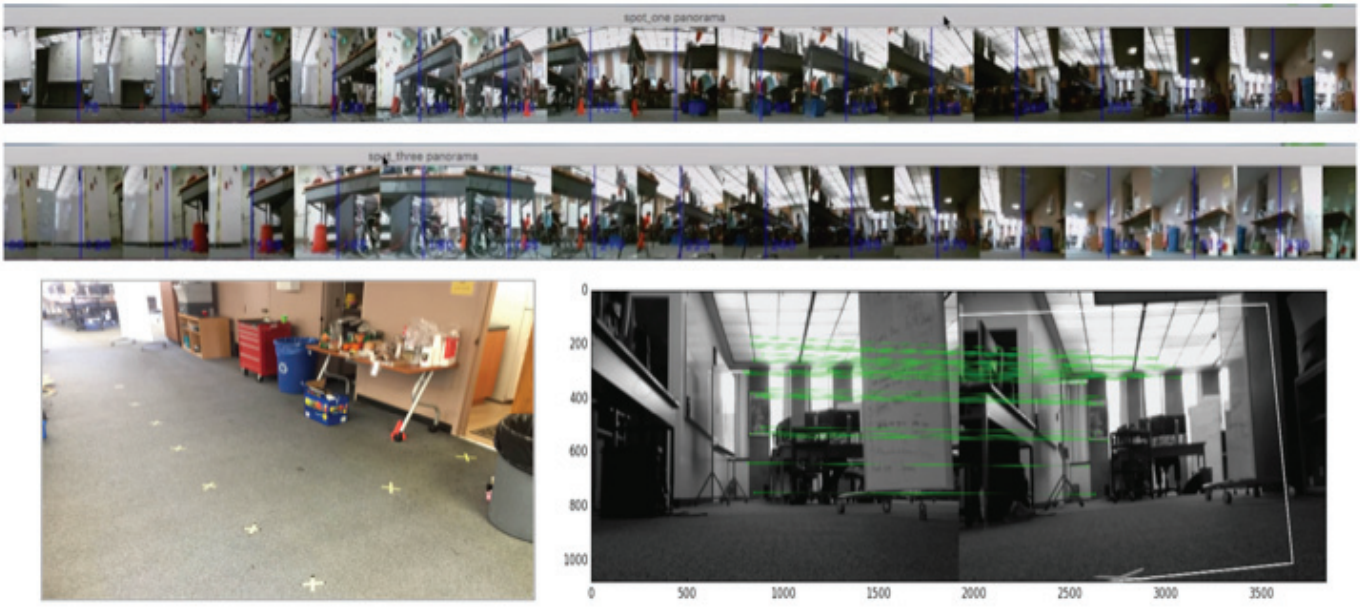

Figure 1. ***(Top)*** *Two slices of the unstitched image-panoramas at the heart of the framework. The images are used as the map from pixel phenomena to location and angle. (**Bottom left**) A human's-eye view of the locations (x's) at which we took panoramas. (**Bottom right**) An example match made via the SURF feature matching and image-warping leveraged from OpenCV.*

Image matching

This last point offers the algorithmic core of the approach. With the graph of panoramas made, image-matching *is* localization. Since images will not match perfectly, match scores are computed against the subset of the images a user wishes to consider – perhaps all of them (for a small environment) or perhaps only those neighboring the most recent localization results, e.g., when filtering.

To determine an appropriate *default* approach for image-matching, we investigated several provided by OpenCV:

1. COLOR, color-histogram matching

2. SIFT, Scale-Invariant Feature Transform (Lowe 2004)
3. SURF, Speeded-up Robust Features (Bay et al. 2008)
4. BOW, using bag-of-words feature-indexing (Nister and Stuwenius 2006).

The first three, color-histogram-matching, and SIFT- and SURF-feature matching are longstanding approaches to comparing images. Figure 1 (bottom right) shows an example match of a panorama's image (at left) with a novel image (right) via SURF feature correspondence. Here, and in the framework, we defined the best match to be the one with the highest number of inlier individual features matches after the geometric adjustment known as homography checking, which seeks to find an aligning transform of one image relative to the other. Bag-of-words approaches operate differently, indexing into a full, pre-processed database (map) of features to find the best-matching location. In our deployment, we found that the bag-of-words approach, at the relatively low resolution of 120x90, worked both quickly and accurately, and we used that algorithm as the default. Such vision-based details can be considered – or not – depending on the goals of the AI/Robotics experience.

Imperfect is perfect, pedagogically speaking

Figure 2 shows the accuracy (in both angle and translation, i.e., the graph-node best matched) along with the run time for a full matching in our 200-image database.

Angle (fraction of correct matches)				
	SURF	SIFT	Color	BOW
80x60	0.776	0.794	0.830	0.807
120x90	0.780	0.798	0.830	0.789
160x120	0.776	0.798	0.825	0.821
320x240	0.803	0.807	0.830	0.825
640x480	0.821	0.825	0.843	0.821
800x600	0.816	0.816	0.825	0.857
Position (fraction of correct matches)				
	SURF	SIFT	Color	BOW
80x60	0.480	0.619	0.330	0.480
120x90	0.547	0.561	0.336	0.565
160x120	0.538	0.565	0.332	0.502
320x240	0.511	0.556	0.332	0.498
640x480	0.516	0.570	0.336	0.507
800x600	0.498	0.570	0.332	0.529
Runtime (ms)				
	SURF	SIFT	Color	BOW
80x60	52.8	111.4	66.8	58.5
120x90	128.9	208.8	67.7	62.2
160x120	232.6	296.1	66.1	66.5
320x240	844.0	999.2	65.0	87.1
640x480	2637.7	9895.1	61.6	136.3
800x600	11559.9	19141.7	67.7	173.4

Figure 2. *Comparison of angular and positional (translational) accuracy (in % of images successfully matched) and runtimes for the four image-matching algorithms considered.*

The default image-matching, then, strikes a pedagogically attractive balance: it's not so cheap that it can be used without consequence, but it's not so expensive that it dominates students' attention. Similarly, it's accurate enough to help a localization algorithm, but not so accurate that it obviates them. What is more, the fact that neighboring panoramas contain similar images offers precisely the same kinds of ambiguity that AI/Robotics algorithms so successfully resolve, e.g., via Particle Filtering and MCL (Simmons and Koenig 1995), Markov Localization (Nourbakhsh et al. 1995; Thrun et al., 2001) or other Bayesian filtering. This framework, in essence, provides the scaffolding needed to include a webcamera or other visual sensor as part of a student project's overall suite.

The Experience

To test the framework's support for robot localization, a team of three rising sophomores implemented and ran several pure-localization tests, in Python, atop the framework. Though not a true classroom test, as they were also the framework's authors, the students *were* entirely new to robotics, robot localization, and AI before their experiments.

Localization only with a Lego robot

First Monte-Carlo Localization's Bayesian-update step was implemented: with each novel image a new population of pose estimates was computed and with each motion, that population shifted probabilistically. The robot used was a hand-built Lego chassis, controlled by a tethered connection to a desktop computer. The Legos held an onboard USB webcamera (for the framework testing). A second USB webcamera, placed overhead, provided ground-truth observations.

Rather than allowing a continuously varying set of poses, however, our approach was to maintain the probability that the robot was near each of the (node, angle) locations in the map. Although the map thus limited the translational precision of this localization system, it did not limit the angular precision: within each panorama image, we estimated the angle of rotation based on image-feature positions, obtaining precision to within 1-2 degrees.

Extension: Handling Blur

Several novel robot traversals were made through the lab area, and significant motion blur affected some, but not all, of the robot's novel images (the original map images themselves were taken at pauses within each panorama's rotation, so they did not suffer from motion blur). Figure 3

shows exemplar images from several unpaused runs of the robot.

To remedy this problem – without pausing the robot for every image – we used the Laplacian operator across each novel image: high variance in this divergence of the gradient indicate the presence of both edges and non-edge (smooth) regions, i.e., a sharp picture. Thus, we weighted the sensor update of each image according to the variance of the Laplace operator for each image. Again this is one example of sensor-specific weighting that is central to the spirit of many AI/Robotics undergraduate experiences. Such blur-handling is not at all a prerequisite for use of the framework! Rather, it illustrates how the framework makes it possible to use cameras within the same algorithms and estimation frameworks that other sensors perhaps more naturally support.

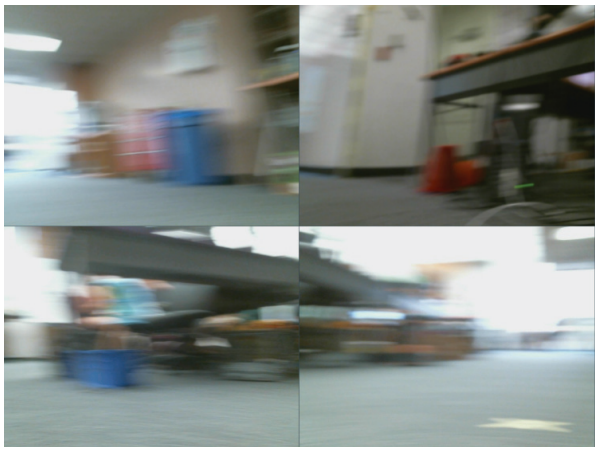

***Figure 3**. Example images from novel (unmapped) robot runs showing the extent of motion blur possible – but not inevitable – under unpaused motion. By weighting images based on a measure of this blur, such images were unweighted during localization.*

Figure 4 shares snapshots of some of the results and of some of the computational interfaces OpenCV makes it easy to create. Not part of the framework per se, students using the library (and their instructors) may determine that other tools for visualizing the localization data better suit their purposes. In each case, the current set of pose estimates appear in the upper left (with mode in cyan and actual angle in green and location in red). The view from the overhead camera providing ground-truth is at the upper right. At the bottom, the best-matched map image is at left, next to the novel image from the robot's camera at right.

In these examples, the lower accuracy of the localization in the first image is due to ignoring motion blur; the far better performance below comes from unweighting the contributions of blurred images.

Localization and Navigation on an iRobot Create

We also tested the framework on a more ambitious indoor localization-and-navigation task -- one we have used in past undergraduate challenges. An iRobot Create sought to navigate to the floor's kitchen, pick up some bagels, and return to its starting point. (Because our focus was the visual support of localization, a waiting accomplice in the kitchen simply placed the bagels on the robot.)

Where this task *did* more deeply integrate the visual localization was in the added use of the accelerometer data from an iPad riding, along with a camera, on the iRobot Create. A Raspberry Pi provided the low-level control and served as a network conduit for the onboard camera. The image-processing continued to happen on the students' desktop, where they found it more convenient to develop and debug their kitchen-trip application.

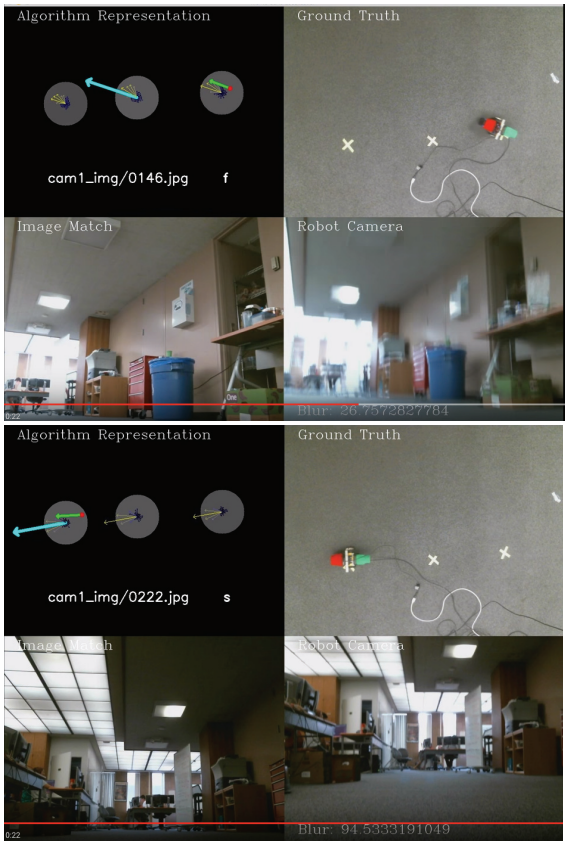

***Figure 4**. (**Top**) Snapshots from two different localization runs, each with three distinct map nodes and 24 distinct angles. By measuring and unweighting blurred images, the bottom run was more successful. (For additional detail, see text.)*

It also Figure 5 provides snapshots summarizing one run, and Figure 6 summarizes the successes and failures across 33 runs of the robot. Here, we compared three image-matching algorithms, with bag-of-words succeeding most

often. We also measured the offset between the robot's final location and its initial location, when it succeeded. The results, called out by Figure 5, highlight the importance of integrating additional sensors.

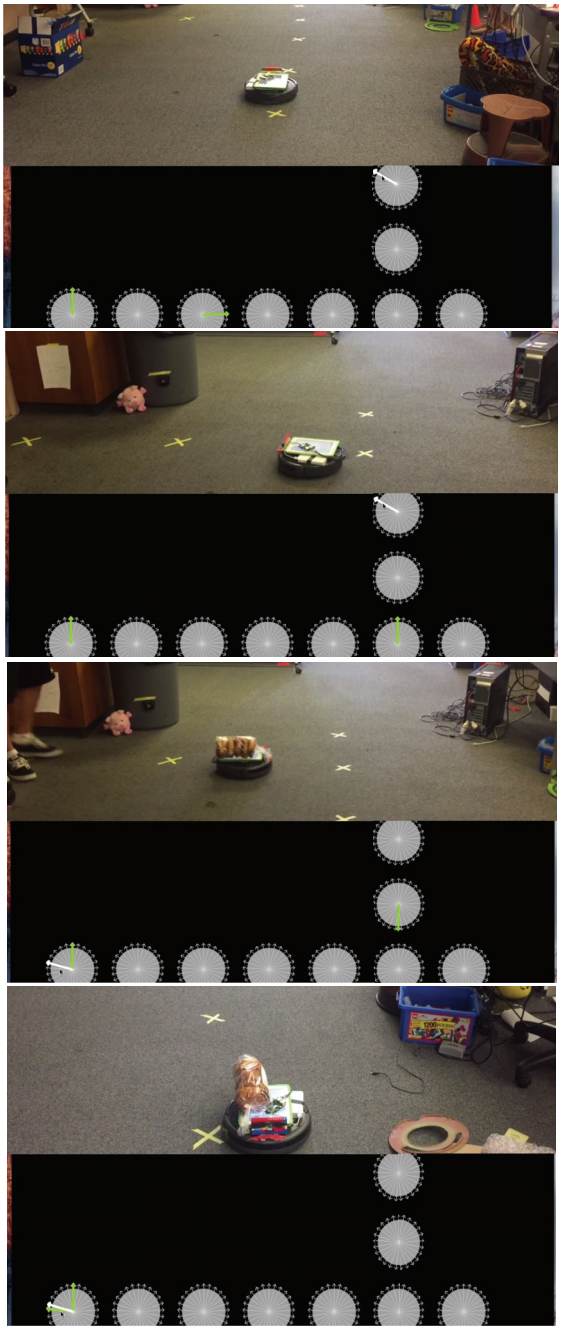

***Figure 5**. Four snapshots including images and pose-estimations from a bagel-fetching task. The Create integrated the framework's vision-based localization along with the accelerometer readings from the iPad on board. A Raspberry Pi rode onboard and provided robot control and the networking to and from a desktop machine, where the image-matching took place.*

Here, the panorama-based visual-localization system is very accurate in estimating *pose angle*, but accumulated translational errors are difficult to correct: neighboring nodes in the map have images too similar in appearance to distinguish between them. Using the accelerometer of an iPad, again developed entirely in Python (using the *Pythonista* application), was as important as the visual localization in making the 25 successful Bagel runs possible!

Overall, the visual-panorama approach to incorporating vision into AI/Robotics projects has a number of strengths: (a) it leverages vision with familiar tools, widely-used libraries, and without recourse to artificial landmarks, (b) it supports the probabilistic reasoning and state-estimation filtering of which the AI/Robotics community if rightfully proud, and (c) it offers a gentle and natural starting point, but one unlimited in the sophistication with which students can pursue insights they gain through their work.

We are excited about toolsets that provide accessible, powerful pathways to composing and understanding our era's core technologies. This paper's framework provides one such foothold for algorithmically uniting vision with undergraduate AI/Robotics. Its codebase and the videos from this work appear at (Zhang 2016). We welcome the refinements that students will doubtless bring to this resource – and to those fields at large – in the future.

	SIFT (320x240)	BOW (320x240)	BOW (800x600)
Trial #	Offsets (inches)		
1	4.5	3.0	2.0
2	MISSED	MISSED	3.5
3	7.5	2.0	2.5
4	MISSED	0.0	2.0
5	3.0	7.5	2.0
6	6.0	5.0	8.5
7	3.0	2.0	3.0
8	MISSED	MISSED	9.5
9	MISSED	5.5	9.0
10	10.0	2.0	9.0
11		MISSED	
12		MISSED	
13		6.0	
Avg.	**5.7**	**3.7**	**5.1**

***Figure 6**. The results (in.) of the final offset of 33 runs of the robot kitchen-trip task. Twenty-five runs overall were successful, with no missed runs by the bag-of-words image-matching approach at a resolution of 800x600 pixels.*

Acknowledgments

The authors gratefully acknowledge the support of the National Science Foundation (CISE REU #1359170) and Harvey Mudd College.

References

Nilsson, N., "A Mobile Automaton: An Application of Artificial Intelligence Techniques," Proceedings International Joint Conference on Artificial Intelligence, Washington, D.C., May 1969. Available as an SRI AI Center Informal Paper, January 1969.

Moravec, H. The Stanford Cart and The CMU Rover, Proceedings of the IEEE, July 1983, pp. 872-884.

Diel, D., DeBitetto, P., Teller, S. Epipolar Constraints for Vision-Aided Inertial Navigation Seventh IEEE Workshops on Application of Computer Vision, 2005. WACV/MOTIONS '05 Vol. 1.

E Johns, S Leutenegger, AJ Davison. Pairwise Decomposition of Image Sequences for Active Multi-View Recognition, arXiv preprint arXiv:1605.08359 May, 2016.

A. Wilby et al, "Design of a Low-Cost and Extensible Acoustically-Triggered Camera System for Marine Population Monitoring,"OCEANS '16 MTS/IEEE, Monterey, CA, Sept. 19-22, 2016.

Ackerman, E. National Geographic Robots Get Intimate With Lions, IEEE Spectrum, 8/12/2013.

Moeys, D. P., Corradi, F., Kerr, E., Vance, P., Das, G., Neil, D., Kerr, D., and Delbruck, T. Steering a Predator Robot using a Mixed Frame/Event-Driven Convolutional Neural Network arXiv:1606.09433 [cs.RO] arxiv.org/abs/1606.09433 June, 2016.

Resnick, M., & Rosenbaum, E. (2013). Designing for Tinkerability. In Honey, M., & Kanter, D. (eds.), Design, Make, Play: Growing the Next Generation of STEM Innovators, pp. 163-181. Routledge. (Tinkerability is a central design principle for many AI/Robotics projects and experiences, just as for Scratch.)

Bradski, G., Learning OpenCV, O'Reilly Media, 2013.

Nister, D. Stewenius, H. Scalable recognition with a vocabulary tree. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), pp. 2161-2168.

Lowe D. G., "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.

Bay, H., Ess, A., Tuytelaars, T., Van Gool, L. "SURF: Speeded Up Robust Features", Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346--359, 2008.

Simmons, R. and Koenig, S. Probabilistic robot navigation in partially observable environments. In Proc. of the International Joint Conference on Artificial Intelligence, 1995.

Nourbakhsh, I., Powers, R., and Birchfield, S. DERVISH: An office-navigating robot. *AI Magazine*, 16(2), Summer 1995.

Thrun, S., Fox, D., Burgard, W., and Dellaert, F. Robust Monte Carlo localization for mobile robots, Artificial Intelligence Volume 128, Issues 1–2, May 2001, Pages 99-141.

Zhang, A. X. zhangxingshuo.github.io/monte-carlo, acc. 12/1/16