Tools for Richer Crowd Source Image Annotations

Joshua Little, Austin Abrams, Robert Pless
Washington University in St Louis
{jdl3|abramsa|pless}@seas.wustl.edu

Abstract

Crowd-sourcing tools such as Mechanical Turk are popular for annotation of large scale image data sets. Typically, these annotations consist of bounding boxes or coarse outlines of objects, in order to keep the interface as simple as possible and to respect browser constraints. However, as most browsers now contain functionality to quickly process images and render shapes to the browser through JavaScript, better annotations can feasibly be generated through the browser given an easy-to-use interface. In this paper, we develop a suite of annotation tools for high-fidelity object contouring and 3D pose working within the limitation that, to be accessible to most Mechanical Turk users, the tools must be available through browsers with no plug-ins or extra downloads. We show comparative results exploring the annotation accuracy relative to existing annotation tools.

1. Introduction

Crowd-sourcing is quickly emerging as the most efficient way to build large datasets with useful labels: the ImageNet database contains a hierarchy of images for all English nouns [4], the CORE dataset describes objects by their respective components [7], and the classic LabelMe annotation tool has been used to segment the objects in over 60,000 images [15]. Each of these datasets made use of crowd-sourcing tools to quickly and cheaply annotate many images with useful labels.

Amazon's Mechanical Turk [2] is a popular resource to harness human intelligence for simple but large-scale annotation tasks, where workers are paid a small amount of money (on the order of pennies per task) to perform a task where a human excels, but computers cannot yet perform reliably.

Mechanical Turk is widely-known as a valuable but arduous tool. Endres et al. [5] report that Mechanical Turk can be "tricky to tame" in that users very rarely read the instructions, there can often be a language barrier, and that many people do not want to spend the time to do a good job.

Figure 1. In this paper, we present web interfaces for Mechanical Turk workers to generate high-fidelity annotations for imagery taken from outdoor cameras. Above, we show example annotations from our car orientation interface. Here, users explicitly define the position, pose, and scale of cars in a surveillance video by manipulating a 3D wireframe model of a canonical car.

This paper presents the viewpoint that by designing easy-to-use interfaces, Mechanical Turk workers can generate richer annotations and labels than previously gathered. To respect the constraints of the user base, all of the tools were written in JavaScript, making use of HTML5 libraries. This removes the need for external plugins that limit the pool of potential Mechanical Turk workers.

We explore this viewpoint with three contributions in this paper:

- An implementation adaptive "livewire" that we release as an open-source JavaScript package, partially automating the accurate contouring of objects in most natural images,
- A 3D wire-mesh rendering tool and interface that per-

mits annotating objects with pose, scale, and location, and

- analyses of the relative accuracy of the annotations with, versus without, these tools.

2. Related Work

The benefits and challenges of Mechanical Turk have been discussed extensively over the last few years. In general, previous work suggests that extreme care be taken when designing an interface intended for arbitrary users. Khanna et al. [11] noted that even after translating an image annotation interface and its instructions from English into the local language for a user study, not one of the participants could draw a bounding box correctly. However, they found that simplifying task instructions and improving the interface increased the annotation quality dramatically. They also suggest using an initial training and validation stage before allowing a user to submit annotations. Heer and Bostock [9] agree that using a validation phase produces substantially better results, and show that Mechanical Turk experiments can reproduce results from previous formal user studies.

In this paper, we develop interfaces to annotate cars in surveillance imagery. Previous datasets contain bounding rectangles and contours of entire cars [1, 6] and their components (e.g., tire, door) [7, 15], in addition to pixel-wise car segmentation [12, 14]. However, there have been few efforts to annotate images with vehicular 3D pose estimation. Perhaps the closest-related dataset comes from Viksten et al. [17], which contains many views of three toy cars under pose variations. This paper presents an open-source tool to collect a pose-annotated dataset "in the wild".

In [3], the authors introduce the Humans in 3D dataset, which contains 3D estimates of human pose. In generating the dataset, a user marks the keypoints from which 3D pose is automatically extracted via the method described in [16]. Although similar in many respects, our 3D pose estimation interface allows a user to directly specify the pose parameters, rather than infer them from keypoints. While we have not yet explored this question, forcing annotations to be consistent with a model may prevent problems from ambiguous or infeasible key points.

In this paper, we present an easy-to-use tool for segmentation of an image. Other interfaces exist for generating ground-truth segmentation of an image into one or more regions, most notably LabelMe [15]. However, our interface is novel in that it uses fine-grained, pixel-tight contours of objects, rather than large, polygonal boundaries.

3. Intelligent Scissors Segmentation

These high-fidelity boundaries are more informative of the shape of the object in question, but little work has been done to encourage users to provide such a boundary. This is likely because specifying pixel-level irregularities in a shape is time-consuming, while denoting a rough polygonal boundary takes less time and is therefore more appropriate for human intelligence tasks in which the worker may grow impatient.

(a)

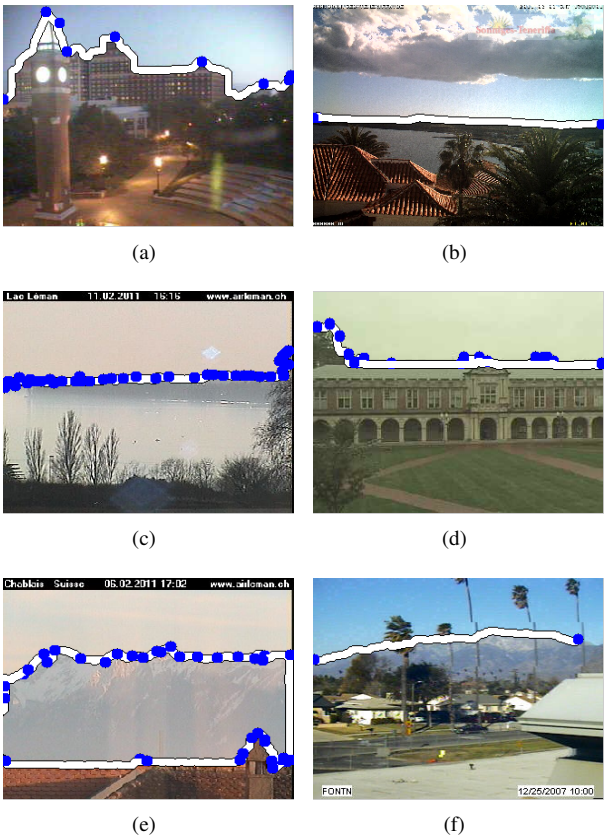
(b)

Figure 2. The Intelligent Scissors interface. (a) The user clicks a point and moves the mouse to specify a contour that separates the sky from the rest of the image. (inset, (b)) The contour automatically snaps to similar-contrast edges in the image. See the supplemental material for a video of the interface in action.

In order to alleviate time concerns, we make use of the popular Intelligent Scissors implementation [13] to automatically suggest contours that fit high-contrast boundaries. Using Intelligent Scissors allows workers to draw a pixel-tight boundary for the area in question, while expending only as much effort as it would take to draw a loose polygonal boundary.

Intelligent Scissors is an interactive optimization that fits a "live wire" contour to some portion of the image. The optimization finds the shortest path from a source pixel (where the user clicked on an image) to a destination pixel (the current mouse position) within a graph where each pixel is a node with edges to its eight neighbors, and edges are weighted by gradient magnitude. Live wire tools perform this optimization interactively, showing the user where a contour would be before accepting it (i.e., by clicking the mouse).

As a test case, we asked workers to delineate the skyline in a webcam image. This task is especially relevant for our interface, because skylines typically contain noisy boundaries (such as edges along a mountainside) and sharp discontinuities. We anticipate that this interface will also be



(a) (b) (c) (d) (e) (f)

Figure 3. Results from the Intelligent Scissors interface. Each image shows the user-provided contour (in white), and the places where a user clicked on the image to lock the previous contour in place (shown as blue dots). Users typically do a good job segmenting sky from non-sky regions, but there are some common failure cases: in (d), the user correctly segmented the sky but doubled back in the opposite direction; in (e), the user mistakenly segmented the mountain instead of the sky; in (f), the user did not complete the task or account for the treeline.

(a) LabelMe [15]

(b) Our results

Figure 4. A comparison of our results to the popular LabelMe annotation tool. The LabelMe contour (a) was generated by 113 mouse clicks, while the Intelligent Scissors contour (b) was generated by 56 mouse clicks by a Mechanical Turk worker.

suitable for other tasks, such as segmenting trees from an image (as was performed in as a first step in [10]) or extending the LabelMe interface [15] with higher-resolution segmentation.

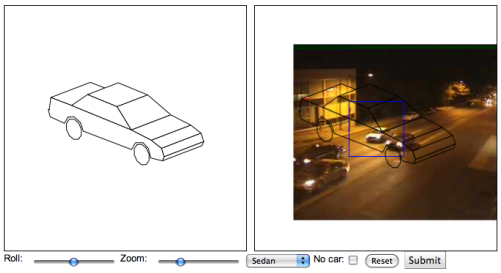
As shown in Figure 2, the appearance of haze and fog will lower the contrast between the sky and not-sky regions. In this case, favoring only high-contrast edges is undesirable, and so we added the additional weight that favors regions with similar contrast as previous boundaries. This allows us to easily prefer lower-contrast boundaries that coincide with the skyline (such as the left inset in Figure 2), rather than high-contrast boundaries.

In our experiments, we found that almost all workers performed the task correctly. Figure 3 shows a few examples from the sky segmentation task. In fact, when compared against more typical results from the popular LabelMe annotation tool [15], we found that users give more precise results, with less overall work (in terms of number of mouse clicks); see Figure 4 for details. For some applications, either segmentation might suffice, but only the Intelligent Scissors interface offers the pixel-level precision to measure (for example) the growth of a tree over time.

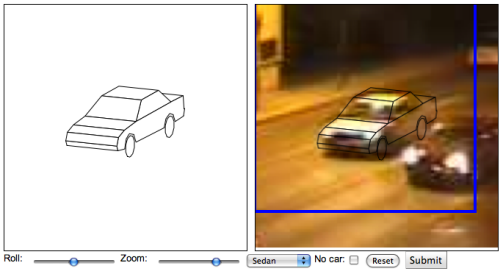
3.1. Implementation

As is the case with all Mechanical Turk jobs, the interface has to be responsive. Workers are often not willing to wait for a task if it is taking too long, especially since they're only paid pennies for each task. This presents a challenge for the intelligent contour interface, because the JavaScript interpreter is relatively slow, single-threaded, and does not have mature support for complex image processing tasks.

Intelligent Scissors makes use of Dijkstra's algorithm to compute the shortest path tree from the path starting point to all pixels in the image. On standard browsers for relevant image sizes, this requires on the order of 10 seconds to complete the shortest path tree, and informal experimentation showed that this was too slow to be feasible. One alterna-



(a)



(b)

Figure 5. Screenshots of our pose estimation interface. (a) The user is given two images: a wireframe car model (left), and the webcam image, with a blue rectangle defining the car to be labeled. (b) The user drags the wireframe car model to get an estimate of vehicle pose, and updates the zoom level and translation. See the supplemental material for a video of the interface in action.

tive is to compute Dijkstra's algorithm for a limited number of steps, but then the user ends up "hunting" for the limits on the image of where the good path is defined. We only found good performance when implementing the algorithm with the JavaScript extension library Web Workers [18]. This library allows a limited form of multi-threaded processing through a message passing paradigm.

This allows our implementation, from the users perspective, to be simultaneously growing the Dijkstra's shortest path tree, while listening for mouse events and drawing the shortest path. Despite the (slight) performance hit in the computation of the shortest path algorithm, Mechanical Turk workers commented that they found this interface more intuitive and engaging.

4. 3D Pose Estimation Interface

In this section, we introduce an interface for 3D pose estimation, as well as some experiments and methods to validate the quality of the data that comes from arbitrary Mechanical Turk workers. As a test case, we consider vehicular pose estimation from many traffic cameras in urban environments. The goal is to get arbitrary users to annotate images describing where cars appear in the image, as well as their orientation and scale. From a vision standpoint, these annotations are more valuable than standard bounding boxes or loose polygonal boundaries, because they capture high-resolution boundaries and pose parameters of objects in a scene.

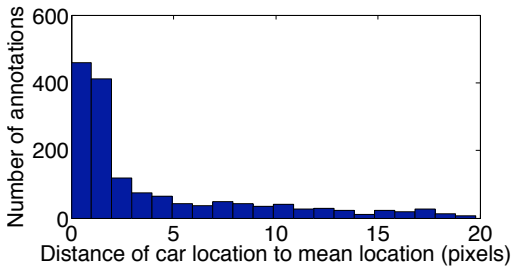


Figure 6. For each car that was labeled by more than one worker, we show the deviation of car position in pixels. This histogram omits the 8 percent of annotations that have a deviation larger than 20 pixels.

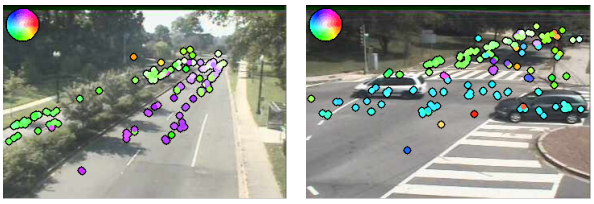


Figure 7. Visualizing car pose annotations. In each plot, the car's pose is shown by an hue-saturation-value scatter plot, where the hue is the projected direction of the car, and the saturation is the scale. Best viewed in color.

4.1. Interface

Our first task is to find out where cars are in the image. We developed a simple interface for a worker to click on all the cars in the image, as a first estimate of car location. These locations are then used to create tasks to infer 3D pose.

In total, there are six degrees of freedom that we want the user to specify: two for position, three for rotation, and scale. We went through several iterations before settling on the pose estimation interface shown in Figure 5. Through this interface, a worker will be given a pre-defined 3D polygonal model of an object and fit that model to an object in the image. Dragging the mouse on the left view changes the pitch and yaw of the model. As completely free rotation is difficult to control with a mouse, roll is instead controlled by a slider bar (although we note that workers seldom use this slider; cars are upright in the image). Dragging the mouse on the right view translates the image, and the image can be zoomed in and out using a slider. Throughout this process, the wireframe model rotates but never changes scale. We chose to scale the image rather than the model because cars in the image can be quite small, and it is typically easier to align a model when it is large and centered.

Figure 8. In addition to annotating car pose, we asked a separate set of workers to create bounding boxes of the selected object. These images show representative annotations from the bounding box annotation (left) and car pose annotation (right) tasks. Although both results cover the object in question, the proposed interface forces users to provide tighter boundaries than the bounding box case.

We used a collection of six polygonal models (sedan, van, pickup, SUV, hatchback, bus) created by hand to cover the major classes of vehicles found in urban environments. The meshes are supplied as a collection of vertices and edges in an OBJ file, a common 3D polygonal model format. Due to the inherent inter-class variation of vehicle shape (i.e., not all sedans look the same), we kept the models simplistic so as not to frustrate the user when a model doesn't exactly line up with the image.

In order to perform this task, we made workers go through a small illustrated tutorial first (as was advised in [11]). In an original trial run of the interface, we simply had a small bulleted list of instructions at the top of the page. We received many submissions that were not rotated or scaled, suggesting a lack of understanding of the interface, or that the users didn't read the instructions, which [5] suggests is a very common occurrence. So, we created a simple multiple-choice validation quiz to give users an idea of the results we expected. After we required this test, submission quality and consistency greatly improved.

As an illustrative experiment, we have 1800 annotations from 258 images taken from many Washington D.C. traffic cameras in late August 2008. These images cover many traffic scenarios (busy streets, intersections, downtown locations, etc.) and times of day.

4.2. Results

In general, we found that users do well annotating cars with 3D pose estimates. Quantitatively, workers tend to strongly agree on car position (the centroid of each car), as shown in Figure 6. Also, Figure 7 shows that users tend to agree on car orientation and scale, although there may be an occasional 180° disagreement. A more qualitative assessment is given in Figure 8, which compares the results from the car orientation interface and a more standard bounding box interface.

The interface allows the use of arbitrary 3D models, so it could also be used to estimate the pose of other objects, such as traffic lights, lane markings, and pedestrians (shown in Figure 9).

Figure 9. Our interface allows pose estimation of arbitrary 3D meshes. In this example, we use the "3D man walking" model from the Google 3D Warehouse [8] to label pedestrians.

We argue that directly estimating 3D pose is a natural and effective way to annotate images. Humans are particularly talented at inferring 3D structure even from single image, and this interface harnesses that ability. Figure 10 shows some successful and unsuccessful annotation results. These results suggest that even if a car is partially occluded (such as the white car in Figure 1) or is not entirely within the field of view (such as the car labeled in Figure 10(e)), humans have a natural and consistent understanding of how the car fits into the 3D world.

5. Conclusions

In this paper, we introduce two interfaces intended to gather high-fidelity annotations from workers in crowdsourcing applications such as Amazon's Mechanical Turk. Our Intelligent Scissors interface allows users to quickly and precisely delineate pixel-tight boundaries for objects, and our 3D pose estimation interface allows arbitrary users to provide 3D annotations of objects in a scene.

Directly estimating the 3D pose is a novel technique that is approachable from novice users and gives very robust results. However, there are some obvious drawbacks to our system. Because we aimed to keep the interface as simple as possible, we don't let the user specify perspective transforms on the 3D mesh. In practice, this is not an issue because the objects we label occupy a narrow part of the visual range and thus are well-approximated by parallel projections. Also, the system requires a 3D mesh, and some classes of objects aren't well-approximated by predefined meshes. For example, labeling the pose of streets, buildings, and trees would be difficult because they occupy many distinct shapes. In future work, we plan to extend our interface to include parametric changes in object shape (e.g. trunk diameter, tree height, etc.).

These interfaces open the doors for future work in studying object detection and pose estimation for scenes "in the wild". Our segmentation interface offers a simple method

(a) (b) (c) (d) (e)

Figure 10. Comparing annotations gathered from multiple users, with a few common failure modes. (a)-(c) Accurate and consistent car pose annotations. (d) The users did not agree on whether the car was moving toward or away from the camera. (e) The users did not agree if the car was a van or SUV.

to denote pixel-wise boundaries of objects, and would be useful for a variety of computer vision tasks including object detection, scene classification, and providing fine-resolution ground-truth functional annotations of a scene. Furthermore, we present a novel method to generate 3D pose estimates in a variety of objects in real scenes. Such annotations could be used as a first step for pose estimation algorithms or exploring complex behavioral changes and traffic patterns in large-scale camera networks. We anticipate that these annotations will improve object detection and pose estimation algorithms, which are classically trained with bounding boxes and loose polygonal boundaries.

This code will be released as an open-source tool to encourage higher quality annotations across all crowd-sourced projects.

References

[1] S. Agarwal, A. Awan, and D. Roth. Learning to detect objects in images via a sparse, part-based representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1475–1490, 2004.

[2] Amazon Mechanical Turk. http://www.mturk.com.

[3] L. Bourdev and J. Malik. Poselets: Body part detectors trained using 3D human pose annotations. In *Proc. IEEE International Conference on Computer Vision*, 2009.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[5] I. Endres, A. Farhadi, D. Hoiem, and D. A. Forsyth. The benefits and challenges of collecting richer object annotations. In *IEEE CVPR Workshop on Advancing Computer Vision with Humans in the Loop*, 2010.

[6] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results. http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html.

[7] A. Farhadi, I. Endres, and D. Hoiem. Attribute-centric recognition for cross-category generalization. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[8] Google 3D Warehouse. http://sketchup.google.com/ 3dwarehouse/.

[9] J. Heer and M. Bostock. Crowdsourcing graphical perception: Using Mechanical Turk to assess visualization design. In *ACM Conference on Human Factors in Computing Systems*, 2010.

[10] N. Jacobs, W. Burgin, R. Speyer, D. Ross, and R. Pless. Adventures in archiving and using three years of webcam images. In *IEEE CVPR Workshop on Internet Vision*, June 2009.

[11] S. Khanna, A. Ratan, J. Davis, and W. Thies. Evaluating and improving the usability of Mechanical Turk for low-income workers in India. In *ACM Symposium on Computing for Development*, 2010.

[12] M. Marszalek and C. Schmid. Accurate object localization with shape masks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[13] E. Mortensen and W. Barrett. Intelligent scissors for image composition. In *SIGGRAPH*, 1995.

[14] A. Opelt, A. Pinz, M. Fussenegger, and P. Auer. Generic object recognition with boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:416–431, March 2006.

[15] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. LabelMe: a database and web-based tool for image annotation. Technical report, MIT AI Lab, 2005.

[16] C. J. Taylor. Reconstruction of articulated objects from point correspondences in a single uncalibrated image. *Computer Vision and Image Understanding*, 80(10):349–363, October 2000.

[17] F. Viksten, P.-E. Forssén, B. Johansson, and A. Moe. Comparison of local image descriptors for full 6 degree-of-freedom pose estimation. In *IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009. IEEE, IEEE Robotics and Automation Society.

[18] Web Workers. http://whatwg.org/ww.