Efficiently Scaling Up Video Annotation with Crowdsourced Marketplaces

Carl Vondrick, Deva Ramanan, and Donald Patterson

Department of Computer Science
University of California, Irvine, USA
{cvondric,dramanan,djp3}@ics.uci.edu

Abstract. Accurately annotating entities in video is labor intensive and expensive. As the quantity of online video grows, traditional solutions to this task are unable to scale to meet the needs of researchers with limited budgets. Current practice provides a temporary solution by paying dedicated workers to label a fraction of the total frames and otherwise settling for linear interpolation. As budgets and scale require sparser key frames, the assumption of linearity fails and labels become inaccurate. To address this problem we have created a public framework for dividing the work of labeling video data into micro-tasks that can be completed by huge labor pools available through crowdsourced marketplaces. By extracting pixel-based features from manually labeled entities, we are able to leverage more sophisticated interpolation between key frames to maximize performance given a budget. Finally, by validating the power of our framework on difficult, real-world data sets we demonstrate an inherent trade-off between the mix of human and cloud computing used vs. the accuracy and cost of the labeling.

1 Introduction

Sorokin and Forsyth [1] made the influential observation that *image* labeling can be crowdsourced at low costs through platforms such as Amazon's Mechanical Turk (MTurk). This approach has revolutionized *static* data annotation in vision, and enabled almost all large-scale image data sets collected since then to be labeled [2–4]. Contemporary computer vision research has subsequently demonstrated the value of massive data sets of labeled images such as the results from ImageNet[2], LabelMe [3], and TinyImages [5].

The same does not hold true for video despite a corresponding abundance of data, such as that from webcams and public-domain archival footage [6]. We believe that this is due to the *dynamic* nature of video data which makes frame by frame labeling necessary but inefficient for manual labor. Inspired by popular successes such as [7–9], we focus on cost effective video annotation with MTurk. We show the results of a year's worth of experiments on learning how to use MTurk to effectively label video. This has resulted in our release of *vatic* (Video Annotation Tool from Irvine, California), the first open platform for monetized,

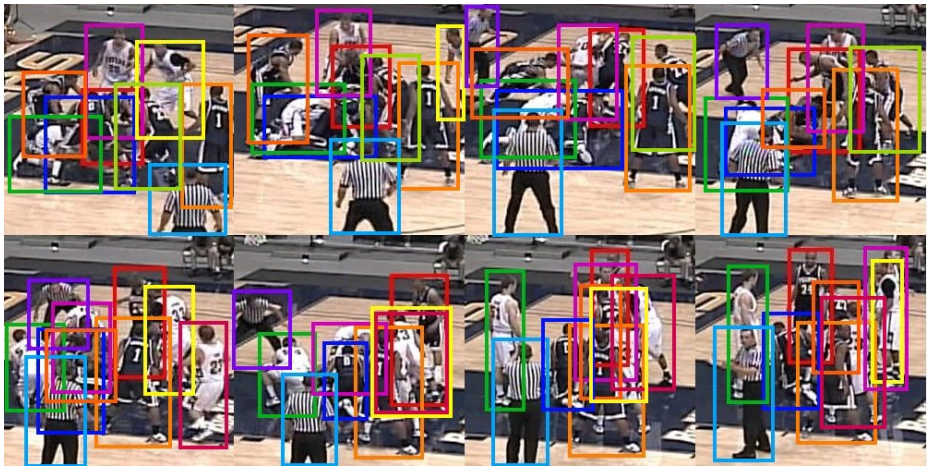


Fig. 1: An example of the difficult problem that our interactive system addresses. The red boxed player becomes totally occluded while many players quickly change pose from standing to a prone position. The referees commonly enter and leave the scene. The ball exists in the pile of people, but even a state-of-the-art vision algorithm is unable to determine its position.

crowdsource video labeling, and a set of "best practices" for creating video-labeling tasks on a crowdsourced marketplace. Our hope is that our findings will spur innovation in the creation of affordable, massive data sets of labeled video.

The contributions made in this paper are motivated by our desire to uncover best-practices for monetized crowdsourced video labeling. In section 2, we present insights into the design of a user-interface in which workers track a single object through a continuous video shot (to solve the problem in Fig.1). In section 3, we analyze trade-offs particular to balancing computer and human effort in video annotation by extending work that minimized labeling cost only along the dimension of human effort [8, 10]. Although the "Turk philosophy" is to completely replace difficult computer tasks (such as video labeling) with human effort, this is clearly *not* efficient given the redundancy of video. In contrast to VideoLabelMe [7], we show that one can interpolate *nonlinear* least-cost paths with efficient dynamic programming algorithms based on image data and user annotated endpoints. In section 4, we analyze the total cost of labeling for various combinations of human workers and cloud-computing CPU cycles. Our final contribution is the release of a simple, reusable, and open-source platform for research video labeling.

2 Mechanical Turk

Amazon's Mechanical Turk is an online labor marketplace that connects employers who have small tasks that are difficult for computers but trivial for humans

and workers. Employers create *Human Intelligence Tasks* (HITs) and set their prices (typically very low) before posting them to the Mechanical Turk servers. Workers browse offered jobs and accept those that interest them. Task completion is not guaranteed and is the result of typical market dynamics [11]. Upon employer validation of completed work, Amazon releases escrowed payments to the workers. Our system uses MTurk to locate workers who can annotate video. We create HITs by first dividing a large video into smaller sequences shot by a single camera. This is accomplished using a standard scene recognition algorithm [12]. We require workers to use our user interface to label the video.

2.1 User Interface

Our user interface (UI) is an interactive browser-based video player that guides workers in labeling a single entity in a sequence of video that is rendered by the client in real-time. Just-in-time compiler optimizations make this feasible in JavaScript.

Nonetheless due to the wide range of systems used by workers, we carefully manage frame caching to enable fast response even on low bandwidth connections. When the video player initializes in the client's browser, the UI immediately requests the first frame of the video from our servers. Subsequent frames are asynchronously buffered only until the next key frame. As many MTurk workers will open the task, but not accept it, we minimize bandwidth overhead by not downloading the entire video sequence. When the user presses "Play", the next frame in the buffer will appear on the screen, the previous frame is discarded from memory, and the next frame to be added to the buffer will begin downloading.

Our approach requires that all frames are individually rendered from the video sequences and saved to our server as JPEG encoded files. While this decompression requires significantly more storage, we are able to optimize the server to efficiently serve these JPEG images. Our custom video player removed artifacts introduced by competing implementations such as the Flash video codec. Long load times and compression artifacts initially prevented effective labeling by workers. Providing a JavaScript based solution also enables much wider participation across platforms and without the need for software licensing (as a MATLAB based solution would require).

Our video player (shown in Fig.2) presents the user with a frame from the video sequence and instructs the user to draw a bounding box around a entity selected from a predefined list. In the case of basketball footage that we tested, we requested that all players, referees and the ball be tracked. After drawing the initial box, the video starts to play. The player software will automatically pause on regularly spaced key frames and prompt the user to label again. Workers annotated a single entity over the entire duration of the sequence.

As entities were annotated by multiple workers in parallel, the resulting tracks were merged into the video stream so that subsequent workers labeling the same video, viewed progressively more densely labeled video. Labeling was complete when multiple workers agreed that there were no more entities left to label.

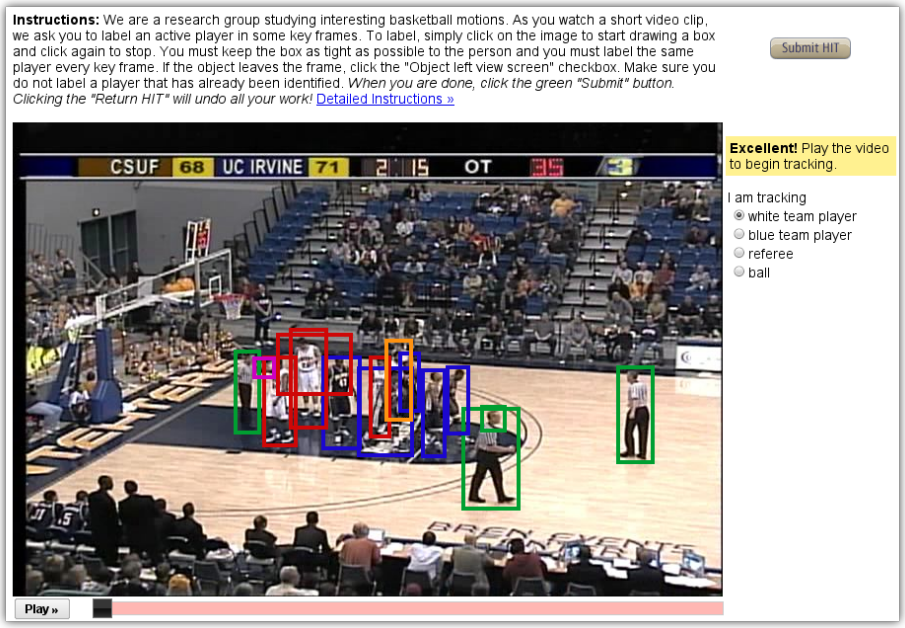


Fig. 2: Our video labeling user interface. All previously labeled entities are shown and the box the user is currently working with is bright orange.

Displaying other workers' labels unintentionally fostered a sense of community engagement that some of workers expressed in unsolicited comments.

> "Maybe it's more bizarre that I keep doing these hits for a penny. I must not be the only one who finds them oddly compelling–more and more boxes show up on each hit." — Anonymous subject

Mechanical Turk does not necessarily ensure quality work is produced. In fact, as a result of the low price of most HITs, many workers attempt to satisfy the HIT with the least amount of effort possible. Therefore it is very important that HITs are structured to produce desired results in a somewhat adversarial environment. One of the key criteria for the design of the UI is to make sure that producing quality work is no harder than doing the minimal amount of work to convince the UI that the HIT is completed. A second important criteria is to build into the evaluation process of a HIT an analysis of the validity of the work. A typical approach is to have multiple workers complete the same task until a statistical test demonstrates consensus on a single answer. A final important criteria is to design the interface so that it is difficult to successfully write an automated bot to get through the UI.

By requiring the user to annotate every key frame or explicitly say there is nothing left to annotate, we reduce the ease with which a worker can just "click-

through" the interface without actually annotating anything. If they have to stop at every key-frame anyway, they may as well do the annotation. Additionally, we automatically rejected any annotations that were degenerate. Typical examples we encountered were to label the first frame and then never move the box, or to have boxes that were very small or very large. It was clear that many of these degenerate tasks were attempted by malicious bots. To ensure the validity of our experiments, we manually validated all labellings, although the production system can test for statistical overlap of multiple workers before accepting the job. In our experiments we found that 35% of the labels had to be discarded as a result of attempts to trick our interface.

2.2 Dense Labeling Protocol

We instruct a worker to label one unlabeled entity in each sequence. After the worker finishes one job, their work is sent back to the server. Once sufficient non-degenerate labels are received to ensure confidence, the data is visually added to the video sequence so that subsequent labeling of the video reflects the new labels. A different worker can then realize that this entity is already labeled and decide to label another entity.

We decided to divide the labor in this manner because it introduces more diversity across workers. Each worker will always be shown a random sequence, so a single worker cannot do all the work for one sequence. Consequently, if a worker provides poor annotations, or shows a systematic bias, they will not taint all entities in a sequence with inadequate performance.

Complex videos of humans typically feature people suddenly appearing from occlusion, and entering and leaving the view frame. If the worker cannot find any unlabeled entities in the initial frame, the worker indicates that all entities of interest are currently labeled. The video player will then advance forward a few frames and ask the user to search for entities that may have entered the frame or appeared from occlusion. This cycle repeats itself until an unlabeled entity is found, or the end of the video is reached. If the video is exhausted and no people have been found, the user implicitly votes to finish the video. After enough people vote, the server stops spawning HITs for this video.

2.3 User Instructions

There are many possible points of failure with non-expert and non-malicious workers. For example, a user may change entities that they are labeling in the middle of the sequence, they may draw bounding boxes which are too large or too small, or they may improperly handle occlusions, entrances and exits of entities from the frame. In order to try to reduce poor labellings, the first time a worker views our HIT they are shown detailed instructions on how to operate our UI. However, workers do not invest time reading instructions because it takes time that they could otherwise be using to complete a different HIT. We attempt to get users started as quickly as possible by showing examples of accepted and

rejected work, alongside detailed text descriptions. Workers can quickly dismiss the written instructions and at any point can return to them.

2.4 Video Server Cloud

We developed our server entirely in Python 2.6 and Cython, both open source and under OSI-approved licenses, so that the system can be deployed on enterprise-scale clusters and compute clouds. While MATLAB is the dominant tool for computer vision work, its license prevents cheap large-scale clustering that such massive amounts of annotation data require. During our experiments, we successfully distributed computation across 20 virtual CPUs using functional programming techniques.

3 Tracking and Interpolation

Vital to our analysis is the ability to properly interpolate between a sparse set of annotations. Our labeling tool requests that a worker labels the enclosing bounding box of an entity every T frames. Offline, we interpolate the object path between these key frames using a variety of algorithms. Because this is done offline, we can afford to use computationally expensive schemes. We define b to be the coordinates of a bounding box:

$$b = \begin{bmatrix} x_1 & x_2 & y_1 & y_2 \end{bmatrix}^T \tag{1}$$

We write b_t for the bounding box coordinate of an entity at time t. Without loss of generality, let us define the keyframe times to be time 0 and time T. We define the interpolation problem to be: Given b_0 and b_T, estimate b_t for $0 < t < T$.

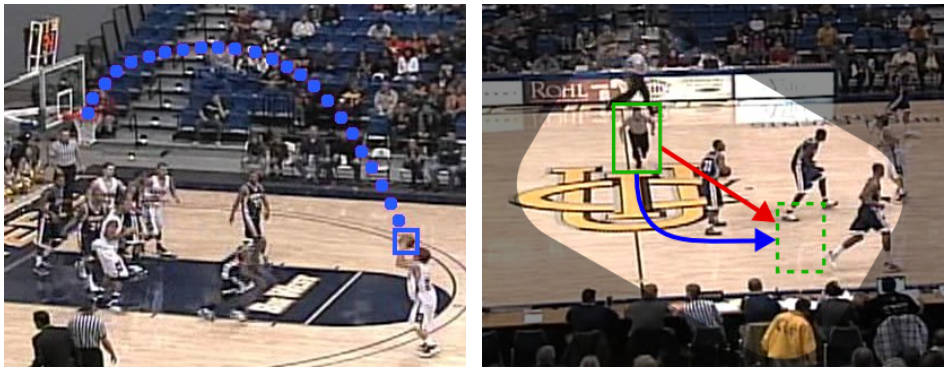
3.1 Linear Interpolation

The simplest approach is linear interpolation:

$$b_t^{lin} = \left(\frac{t}{T}\right) b_0 + \left(\frac{T-t}{T}\right) b_T \quad \text{for} \quad 0 \leq t \leq T \tag{2}$$

[7] makes the astute point that constant velocity in 3D does not project to constant velocity in 2D due to perspective effects. Assuming the tracked entity is planar, they describe a homography-preserving shape interpolation scheme that correctly models perspective projection. However, we found simpler linear interpolation to work well for many common scenes where there does not exhibit much depth variation.

Both of these interpolation algorithms are very efficient since they avoid the need to process any pixel data. However, both assume constant velocity which clearly does not hold for many entities (Fig.3a). We describe a dynamic-programming-based interpolation algorithm that attempts to *track* the location of the entity given the constraints that the track must start at b_0 and end at b_T. While there are many trackers available [13], we chose a simple method so we can focus on the economics of MTurk video annotation.



(a) Basketballs do not move linearly. (b) The error tube on the linear path.

Fig. 3: Nonlinear motion requires more sophisticated interpolation strategies for estimating entity position given the end locations. We employ a visual tracker in the error tube of the linear path in order to find the actual path (in blue) through visual analysis.

3.2 Discriminative Object Templates

To score a putative interpolation path, we need a visual model of the tracked object. We use all the annotated keyframes within a single video shot to build such a model. Let us assume N such keyframes exist, which in turn yield N bounding boxes which contain the entity of interest. Our first approach was to construct an average pixel-based template, and score it with sum-of-squared differences (SSD) or normalized correlation. We also experimented with more sophisticated templates built on histogram of oriented (HOG) [14] features. We found poor results with both.

We believe the suboptimal results arose from the fact that such templates are not designed to find objects in the cluttered backgrounds that we encountered. To compensate for this fact, we extract an *extremely* large set of "negative" bounding boxes collected from the N keyframes, making sure that they do not overlap the entity of interest in those frames. We then attempted to score a putative bounding box by competing an object template with an average background template. We again found poor results, this time due to the fact that our video backgrounds are complex and are poorly modeled with an average template.

Finally, we converged on the approach of learning a discriminative classifier trained to produce high scores on positive bounding boxes and low scores on the negatives. For each bounding box b_n we compute a feature descriptor composed of HOG and color histogram features:

$$\phi_n(b_n) = \begin{bmatrix} HOG \\ RGB \end{bmatrix} \tag{3}$$

We use a RGB color histogram with 8 bins for each dimension. Before extracting features, we resize the image patch at b_n to the "canonical" object size estimated from the average of the N labeled bounding boxes. Given a collection of features along with labels $y_n \in \{-1, 1\}$ identifying them as positives or negatives, we learn a linear SVM weight vector w that minimizes the following loss function:

$$w^* = \text{argmin} \frac{1}{2} w \cdot w + C \sum_n^N \max(0, 1 - y_n w \cdot \phi_n(b_n)) \tag{4}$$

We use liblinear [15], which appears to be fastest linear SVM solver available. For typical size problems, training took a few seconds.

3.3 Constrained Tracking

Let us write the constrained endpoints given by the keyframes b_0^* and b_T^*. We wish to use the template w to construct a low-cost path $b_{0:T} = \{b_0 \dots b_T\}$ subject to the constraints that $b_0 = b_0^*$ and $b_T = b_T^*$. We score a path by its smoothness and the local classifier scores:

$$\underset{b_{1:T}}{\text{argmin}} \sum_{t=1}^{T} U_t(b_t) + P(b_t, b_{t-1}) \tag{5}$$

$$s.t. \quad b_0 = b_0^* \quad \text{and} \quad b_T = b_T^* \tag{6}$$

We define the unary cost U_t to the negative SVM score plus the deviation from the linear interpolation path. In order to reduce the penalty on occlusions, we truncate the cost by α_2:

$$U_t(b_t) = \min\left(-w \cdot \phi_t(b_t) + \alpha_1 ||b_t - b_t^{lin}||^2, \alpha_2\right) \tag{7}$$

We define the pairwise cost to be proportional to the change in position:

$$P(b_t, b_{t-1}) = \alpha_3 ||b_t - b_{t-1}||^2 \tag{8}$$

Note that the constraints in (6) can be removed simply re-defining the local costs to be:

$$U_0(b_0) = \inf \quad \text{for} \quad b_0 \neq b_0^* \qquad \text{and} \qquad U_T(b_T) = \inf \quad \text{for} \quad b_T \neq b_T^* \tag{9}$$

3.4 Efficient Optimization

Given K candidate bounding boxes in a frame, a naive approach for computing the minimum cost path would take time $O(K^T)$. It is well known that one can use dynamic programming to solve the above problem in $O(TK^2)$ by the following recursion [16]:

$$cost_t(b_t) = U_t(b_t) + \min_{b_{t-1}} cost_{t-1}(b_{t-1}) + P(b_t, b_{t-1}) \tag{10}$$

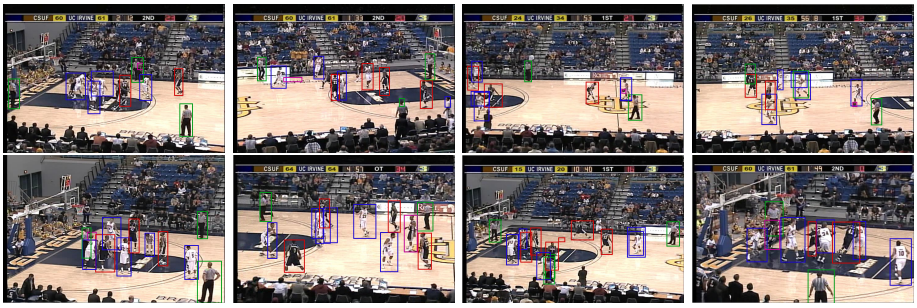


Fig. 4: Example of annotations from MTurk workers on difficult basketball footage. Red boxes are labeled as white team, blue as blue team, green as referee, and purple as the basketball. Worker quality is good, but not perfect.

where $cost_t(b_t)$ represents the cost of the best path from $t = 0$ to b_t. We initialize $cost_0(b_0) = U_0(b_0)$. By keeping track of the argmin, one can reconstruct the minimum cost path ending at any node.

We limit K by restricting the set of putative bounding boxes at time t to lie within some fixed deviation from b_t^{lin}, both in terms of position and scale (Fig.3b). We found this pruning greatly improved running time and accuracy. We also note that the above recursion can be written as a min-convolution [17], allowing us to compute the optimum in $O(TK)$ using distance-transform speed-ups:

$$cost_t(b_t) = U(b_t) + \min_{b_{t-1}} cost_{t-1}(b_{t-1}) + \alpha_2 ||b_t - b_{t-1}||^2 \tag{11}$$

4 Results

We validate our online labeling framework by placing three different data sets of varying difficulty on MTurk. First, we look at "easy" videos of high-contrast entities moving in uncluttered backgrounds with little occlusion. We use YouTube sports footage of athletes performing running drills (as in Fig.5). Second, we look at a "marginally hard" task of annotating basketball players who tend to undergo a fair number of occlusions in quite cluttered backgrounds (as in Fig.1 and Fig.4). Finally, we consider the task of annotating "very hard" entities such as a basketball (as in Fig.3), which is hard to track due to frequent occlusions by players and the existence of large amounts of motion blur relative to its small image size. We use these data sets to examine cost trade-offs between automation and manual labeling as a function of the difficulty of the data. Our labeled basketball video data is unique for its size and complexity, and we plan to make it available to the community for further research on activity analysis.

We employ our previously described user interface to have workers annotate every fifth frame of our video sets, including a two-hour (210,000 frame) basketball game. We use this dense set of MTurk annotations as ground truth. We

Fig. 5: An example of our "sports drill" data set containing nonlinear motion on uncluttered backgrounds.

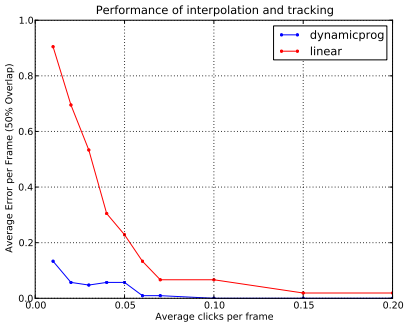
then hold out different intervals of the ground truth in order to determine how well the tracking and interpolation methods predict the missing annotations. We score our predictions with the same criteria as the PASCAL challenge: a prediction must overlap with the actual annotation by at least 50% to be considered a detection.

4.1 Diminishing Returns

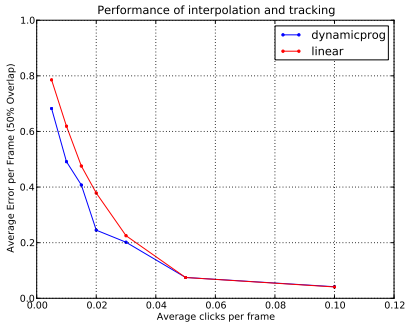
We first confirm our hypothesis that the "Turk philosophy" (human computation is cheap and subsumes automated methods) does not hold for video because it is wasteful for users to annotate every frame. Fig. 6 shows a diminishing returns property in which increased human labeling (x-axis) results in smaller and smaller reductions in error rates (y-axis). Moreover, the rate of the diminishing is affected both by the difficulty of the video (easy, medium and hard) and the choice of interpolation algorithm (linear interpolation in red and dynamic programming in blue). For easy videos, we can achieve 10% error with a user annotation rate of 0.05 clicks per frame. For medium-difficultly videos, we require at least 0.1 clicks per frame regardless of the mode of interpolation. Finally, for difficult videos, we need 0.2 clicks per frame for the best accuracy. Our results suggest that interpolation of any kind can exploit the redundancy in video to reduce annotation effort by an order of magnitude compared to a naive, "brute-force" MTurk approach.

4.2 CPU vs Human Cost

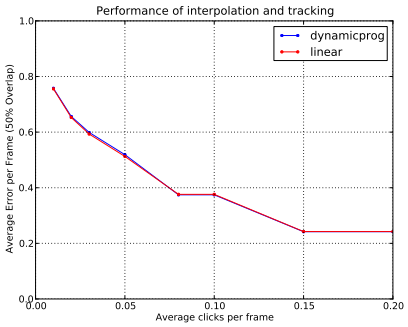
We now consider how to optimally divide CPU effort vs human effort (frequency of annotation) so as to maximize track accuracy. We make three reasonable assumptions: linear interpolation is free in terms of CPU effort; tracking-based interpolation requires a fixed amount of computation effort regardless of annotation frequency; and we can smoothly increase CPU effort from 0% to 100% by choosing to linearly interpolate or use a dynamic programming tracker for each key frame interval. If we wish α CPU effort then we randomly choose to use a dynamic programming tracker by flipping an α-biased coin.



(a) Field drills (easy)



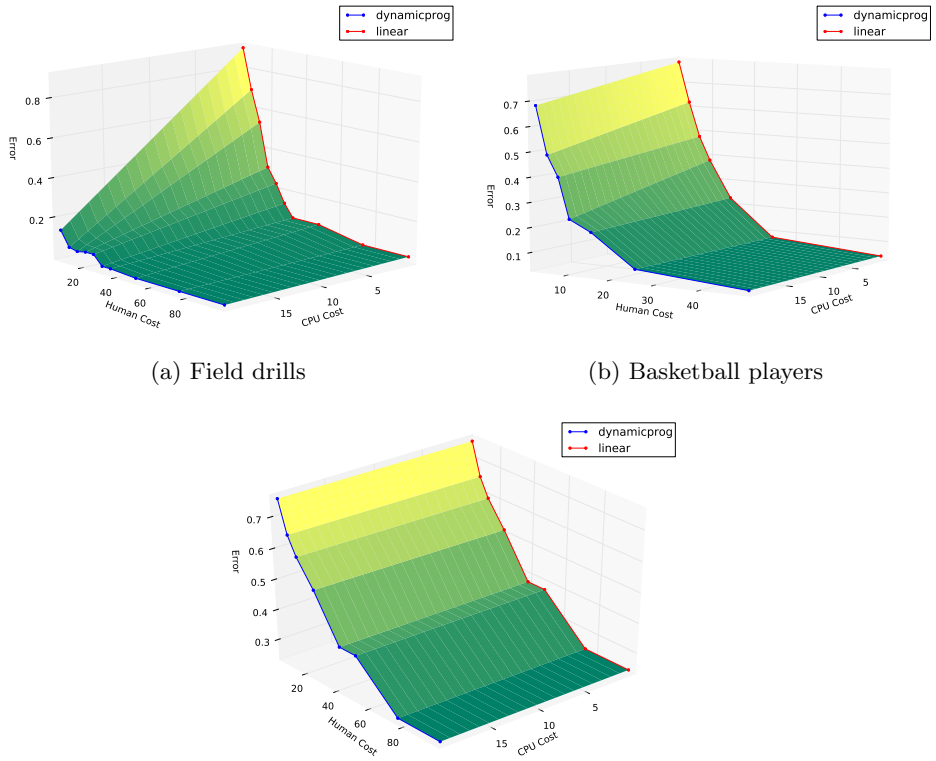
(b) Basketball players (medium)



(c) Ball (hard)

Fig. 6: Performance of dynamic programming and 2D linear interpolation on easy, marginally-difficult, and very-difficult data sets. Dynamic programming excels when features are easily extracted, such as in the field drill. But, dynamic programming performs equally well as linear interpolation when the object is highly occluded (such as a basketball).

We use rates from Amazon's Elastic Compute Cloud (EC2) platform to compute a monetized CPU cost. Our tracking algorithm will take 102 hours on EC2 to fully process a two hour basketball game. At US$0.17 per hour, the tracking costs US$17.34. We compute a monetized human cost by subtracting the amount we paid MTurk workers. We compensated workers US$0.02 per HIT to label every fifth frame, with a total of $483.20 to have MTurk label every object in every frame. We note, however, that such a rate does not appear to be sustainable – many workers loudly complained about this rate and demanded at least $0.10 per HIT. Finally, we compute the tracking error produced by various dollar amounts devoted to human labeling versus CPU usage. Not surprising, as we spend more money, label error decreases.



(a) Field drills

(b) Basketball players

(c) Ball

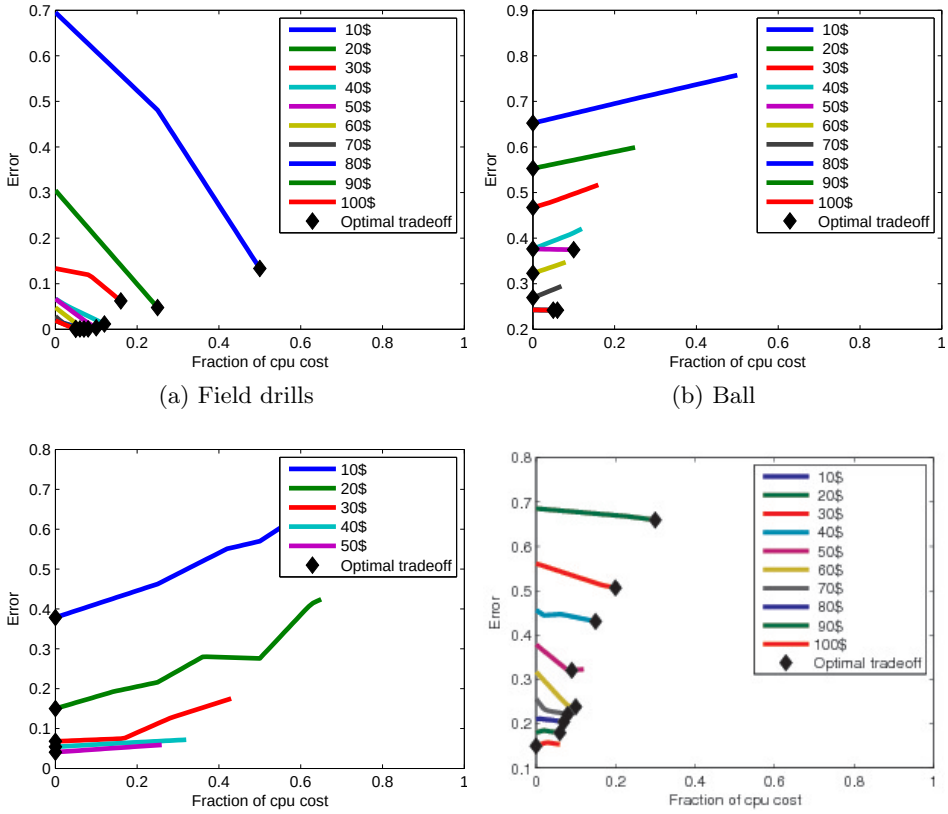
Fig. 7: Cost trade-off between human effort and CPU cycles. As the total cost increases, performance will improve. Cost axes are in dollars.

4.3 Performance Cost Trade-off

We now consider our motivating question: how should one divide human effort versus CPU effort so as to maximize track accuracy given a X$? A fixed dollar amount can be spent only on human annotations, purely on CPU, or some combination. We express this combination as a diagonal line in the ground plane of the 3D plot in Fig.7. We plot the tracking accuracy as a function of this combination for different X$ amounts in Fig.8. We describe the trade-off further in the caption of Fig.8.

5 Conclusion

Our motivation thus far has been the use of crowdsource marketplaces as a cost-effective labeling tool. We argue that they also provide an interesting platform for research on *interactive* vision. It is clear that state-of-the-art techniques are



(a) Field drills (b) Ball

(c) Basketball players (d) Basketball players (alt cost)

Fig. 8: We show the cost trade-off between human effort and CPU cycles for different dollar amounts on different videos. In the "easy" field-drill video (**a**), the optimal trade-off is to maximize CPU usage for a fixed dollar amount. In the "very-difficult" ball footage (**b**), the optimal trade-off is to minimize CPU usage for a fixed dollar amount, essentially reducing to linear interpolation. Our most interesting trade-off results occur for the basketball footage. At our current MTurk rate of 2 cents per shot (**c**), the optimal trade off is to minimize CPU usage and use linear interpolation with frequent annotations. At a proposed MTurk rate of 10 cents per shot (suggested by many annotators) and half the CPU cost (likely achievable in the near future) (**d**), the optimal trade off is to maximize CPU usage for a fixed dollar amount.

unable to automatically interpret complex visual phenomena. Our hypothesis is that by allowing a modest amount of human intervention, one can *can* successfully deploy vision algorithms which incrementally can measure and quantify progress for such difficult scenarios. To demonstrate this, our analysis in this

paper focused on basketball sports footage. We note that there has been relatively little work in this domain, compared to tennis or soccer, perhaps because of clutter and the many occlusions. Indeed, we know of no algorithm that can correctly track the players in Fig.1.

Acknowledgments: Funding for this research was provided by NSF grants 0954083 and 0812428. We thank the thousands of MTurk workers for their participation.

References

1. Sorokin, A., Forsyth, D.: Utility data annotation with amazon mechanical turk. Urbana **51** (2008) 61820
2. Deng, J., Dong, W., Socher, R., Li, L., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: Proc. CVPR. (2009) 710–719
3. Russell, B., Torralba, A., Murphy, K., Freeman, W.: LabelMe: a database and web-based tool for image annotation. International Journal of Computer Vision **77** (2008) 157–173
4. Kumar, N., Berg, A.C., Belhumeur, P.N., Nayar, S.K.: Attribute and Simile Classifiers for Face Verification. In: IEEE International Conference on Computer Vision (ICCV). (2009)
5. Torralba, A., Fergus, R., Freeman, W.: 80 million tiny images: a large dataset for non-parametric object and scene recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence **30** (2008) 1958–1970
6. http://www.moviearchive.org (2010)
7. Yuen, J., Russell, B., Liu, C., Torralba, A.: LabelMe video: Building a Video Database with Human Annotations. (2009)
8. Vijayanarasimhan, S., Grauman, K.: Whats It Going to Cost You?: Predicting Effort vs. Informativeness for Multi-Label Image Annotations, CVPR (2009)
9. Liu, C., Freeman, W., Adelson, E., Weiss, Y.: Human-assisted motion annotation. In: IEEE Conference on Computer Vision and Pattern Recognition, 2008. CVPR 2008. (2008) 1–8
10. Vijayanarasimhan, S., Jain, P., Grauman, K.: Far-Sighted Active Learning on a Budget for Image and Video Recognition, CVPR (2010)
11. Ross, J., Irani, L., Silberman, M.S., Zaldivar, A., Tomlinson, B.: Who are the crowdworkers? shifting demographics in mechanical turk. In: alt.CHI session of CHI 2010 Extended Abstracts on Human Factors in Computing Systems. (2010)
12. Oliva, A., Torralba, A.: Modeling the shape of the scene: A holistic representation of the spatial envelope. International Journal of Computer Vision **42** (2001) 145–175
13. Avidan, S.: Ensemble tracking. IEEE Transactions on Pattern Analysis and Machine Intelligence **29** (2007) 261–271
14. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: CVPR. (2005) I: 886–893
15. Fan, R., Chang, K., Hsieh, C., Wang, X., Lin, C.: LIBLINEAR: A library for large linear classification. The Journal of Machine Learning Research **9** (2008) 1871–1874
16. Bellman, R.: Some problems in the theory of dynamic programming. Econometrica: Journal of the Econometric Society (1954) 37–48
17. Felzenszwalb, P., Huttenlocher, D.: Distance transforms of sampled functions. Cornell Computing and Information Science Technical Report TR2004-1963 (2004)