CrowdSight: Rapidly Prototyping Intelligent Visual Processing Apps

Mario Rodriguez and **James Davis**
Computer Science Department
University of California, Santa Cruz, USA
{mrod, davis}@soe.ucsc.edu

Abstract

We describe a framework for rapidly prototyping applications which require intelligent visual processing, but for which reliable algorithms do not yet exist, or for which engineering those algorithms is too costly. The framework, CrowdSight, leverages the power of crowdsourcing to offload intelligent processing to humans, and enables new applications to be built quickly and cheaply, affording system builders the opportunity to validate a concept before committing significant time or capital. Our service accepts requests from users either via email or simple mobile applications, and handles all the communication with a backend human computation platform. We build redundant requests and data aggregation into the system freeing the user from managing these requirements. We validate our framework by building several test applications and verifying that prototypes can be built more easily and quickly than would be the case without the framework.

1 Introduction

Mobile applications that make use of visual sensing are a promising future afforded by cheap cameras built into nearly every mobile phone. Take a picture with one click, intelligent processing occurs, something useful gets done. Examples of existing applications include business cards scanners that populate a contacts database (BusinessCardReader; BizSnap) and product scanners that search for reviews and lowest prices (SnapTell; RedLaser; AmazonRemembers). We hypothesize that there are many more such applications and services which have yet to be imagined.

These applications are powered by computer vision algorithms that process the images, e.g. the majority of the business cards apps use Optical Character Recognition (OCR). Taking the fully automated approach to building a system with high intelligence requirements involves setting up a development team with the appropriate expertise, spending several months developing the necessary algorithms and models, gathering and analyzing training data, and validating and tuning the models to ensure that the system satisfies user requirements. One problem with this approach is that the primary hypothesis, that users actually want to use the system, is only validated in the final stages of the development process, after considerable development expense.

Another problem with developing fully automated computer vision algorithms is there is often a mismatch between expectations of robustness and what is actually obtainable. For example, experiments with a business cards app revealed that there is considerable variability in the quality of the results obtained, with some of the influencing factors being the lighting quality of the photo and whether the user zoomed in on the area of interest in the business card. In cases of low quality results, the user is tasked with manually entering the information, destroying the usefulness of the service. Unreliable results of this nature often cause the users to abandon the application after a few tries.

Another approach is to employ humans in the computer vision processing loop, reducing development costs and increasing robustness. In recent years, we have witnessed the emergence of platforms which enable the use of human intelligence to be injected into the execution pipeline of applications, Amazon Mechanical Turk being one such service (AMT). However, the overhead to set up and use AMT from a mobile environment is sufficiently high that most application developers have not yet done so. Amazon itself is an exception, having deployed a mobile application that utilizes human labor for identifying products and providing a link for purchasing the product on Amazon (AmazonRemembers).

We introduce CrowdSight, a framework which makes it as easy as possible to quickly prototype these kinds of applications requiring intelligent visual processing and where the input into the system is a single static image. Our system is a web service that takes requests from mobile applications or via email. The visual computation is performed using AMT as a human computation engine. Crowdsight abstracts as much of the processing and communication overhead as possible, in order to simplify quickly prototyping new ideas. A new service is defined on a single webpage and consists primarily of instructions which explain what the human worker is supposed to accomplish. We expose various parameters and presentation options used by AMT, but they can frequently be left in a default state. The service provides for redundant requests and result aggregation, and the "service developer" need only specify the amount of redundancy required to achieve acceptable robustness.

The primary contribution of this paper is a framework for quickly prototyping new mobile applications requiring visual processing. We support this contribution by testing our framework through the development of several new services. We hypothesize that by making it cheap and easy to prototype such applications, the probability of identifying the useful ones will increase simply by trying many of them. We have developed several applications to validate that it is easy and fast to deploy new services with our framework. Our new services include: transcription, business card reader, product search, and people counting. These services are discussed in section 4.

2 Related Work

Development Tools for Human Computation: Research on using human labor for computation is extensive, for a survey see (Quinn and Bederson 2010). Some researchers have sought to raise the level of abstraction involved in developing human computation applications. For example, TurKit is a toolkit for deploying iterative tasks to AMT, with an imperative programming paradigm that uses turkers (AMT workers) as subroutines (Little et al. 2009). The toolkit handles the latency of HITs (AMT Human Intelligence Tasks), supports parallel tasks, and provides fault tolerance. There has also been research on integrating human computation into a database query language such that evaluation of specific predicate operators would automatically expand into HITs which would then be managed by the query processing engine (Parameswaran and Polyzotis 2011). Another example is QuikTurkit, which is a set of tools to decrease latency when calling human computation routines (Bigham et al. 2010). All of these tools are targeted at developers and are complementary to the role of CrowdSight, which is intended to make prototyping a new mobile application as quick as possible.

Mobile Apps using Human Computation: Crowdsourcing has also been used to power mobile applications. Amazon Remembers locates a web link to purchase products the user photographs. mCrowd is an iPhone based mobile crowdsourcing platform that enables mobile users to post and work on sensor-related crowdsourcing tasks (Yan et al. 2009). mCrowd enables users to post tasks to AMT (such as asking turkers to tag an image or to take a picture of something in particular), however, those tasks come from a fixed, predefined set, built into the application. VizWiz is a talking application that enables vision-impaired users to take a picture of a situation in life and ask a question about it to workers in the cloud (Bigham et al. 2010). Many other applications and domains exist. CrowdSight is intended to allow developers to quickly prototype and test new applications and services.

3 System Architecture

CrowdSight is primarily a web application that simplifies creating new services that rely on AMT on the backend. Figure 1 shows an overview of the system. Service developers create new services in the web application. Once a service has been created, end users can utilize it by submitting requests from their mobile devices. End users simply need the

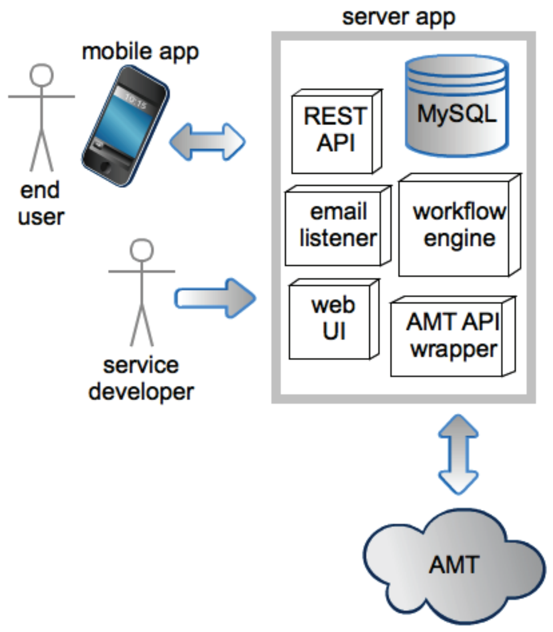


Figure 1: CrowdSight system overview. The system allows mobile end-users to submit requests to the server app using a mobile app. Service developers create new services using the web UI. The workflow engine manages communication with the crowdsourcing platform, Amazon Mechanical Turk (AMT).

ability to email an image from their mobile devices. Alternatively, we have developed a mobile application that provides a more user-friendly experience and additional capabilities. The CrowdSight engine processes requests submitted by the end users; it handles communication with AMT, abstracting away as many details as possible.

3.1 Web Application

Any new CrowdSight service requires both the primary description of what workers are supposed to accomplish as well as a methodology to ensure robustness and quality. Most AMT developers ensure quality through multiple requests and worker result aggregation, which often involves a secondary round of interaction with AMT. CrowdSight provides an abstraction for this process so that the developer does not need to explicitly manage each of these tasks.

Creating a new service: Defining a service in CrowdSight entails filling in a web form with a few parameters and XML specifications of the HIT. Each service has a unique ID which is used to specify which CrowdSight service is being interacted with (whether by email or through the mobile application), as well as a few fields providing a title and describing the service.

AMT requires a set of parameters specified as a list of key-value pairs for defining a HIT. It would be possible to completely abstract this specification from CrowdSight developers for ease of use, however we have found that developers sometimes want to tweak these values after they

```
title:Transcribe text in image
reward:0.05
assignmentduration:3600
qualification.comparator.1:greaterthan
qualification.value.1:25
qualification.private.1:false
```

Figure 2: Service properties definition. A list of key-value pairs containing service specification details. The qualification specifies a required approval rate of at least 25% for potential turkers (AMT workers).

```
<QuestionForm>
<Question>
<QuestionContent>
<Text>Transcribe text in image:</Text>
</QuestionContent>
<AnswerSpecification>
<FreeTextAnswer>
<NumberOfLines>10</NumberOfLines>
</FreeTextAnswer>
</AnswerSpecification>
</Question>
</QuestionForm>
```

Figure 3: Question-Answer example. A HIT (AMT Human Intelligence Task) can be specified declaratively using the format specified by the Question-Answer schema definition.

have an initial version of their new service running. We compromise by providing an example set of default parameters which most developers use initially, however they can also edit the parameters if desired. A sample of basic HIT properties is included in figure 2.

The Question-Answer field is an XML document, an example of which is shown in figure 3, that allows for a declarative definition of the structure and presentation of a HIT. Using XML to specify the task is a compromise of functionality and ease of use. It might be possible to define many services simply by providing a text description of what the worker is supposed to do with the given image. However we want to provide for both simple and more complex services. We provide an example XML task description, and even novice users find it simple enough to locate the text description in this block and edit it for their needs. More advanced users can specify arbitrarily complicated structured input to match their needs. The AMT documentation provides a complete description of the schema, which allows the inclusion of CDATA tags into which HTML may be embedded to control the presentation details of the HIT.

Managing Robustness: Responses provided by turkers have a very high error rate, with 30% error having been suggested as a good rule of thumb (Bernstein et al. 2010). Some sort of error management scheme is needed. Research has shown that averaging answers from many workers often improves the quality of the results (Snow et al. 2008). When the nature of each query is known, it is possible to design custom verification and aggregation methods, such as averaging numerical answers, or testing values against known check-

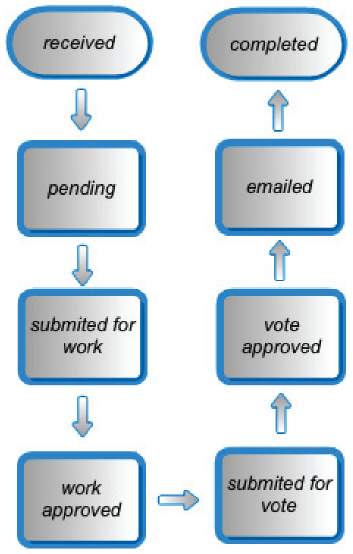
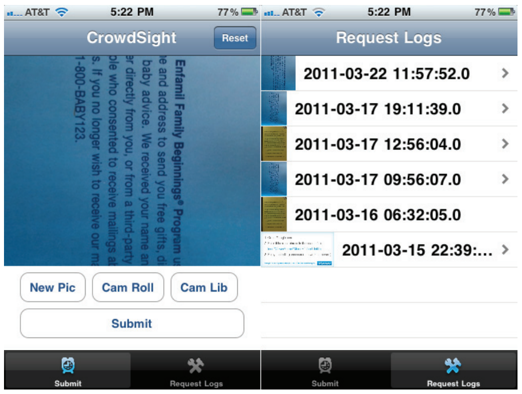


Figure 4: Service request stages. The request is submitted for work and vote to the crowdsourcing platform. Progressing a service request through the pipeline is managed by the workflow engine depicted in figure 1.

sum bits (Davis et al. 2010). We would like our framework to abstract this verification step, however we know nothing in advance about the nature of the data being collected.

In order to generalize the verification step, we implement a strategy of asking for several answers, and then submitting the results to another round of turkers to rate those answers. This is far from optimum, since data specific aggregation might do much better with fewer requests. However our goal is to be general and simple, as opposed to maximally cost effective. When a service is defined, the developer can specify the number of turkers who should perform the same task. The developer also specifies the number of ratings each response should receive. Each rating task presents the original request image and the response, and asks the turker to rate the response on a scale of 1 (wrong) to 5 (perfect). Finally, the ratings for a given response are averaged and the response with the highest average rating is returned to the user. For example, if the developer asks for 5 responses and 3 ratings per response, the system will submit a total of 20 tasks to AMT for each image submitted by a user. We have found that this methodology allows developers to increase robustness easily and arbitrarily across a wide range of tasks, without having to produce an aggregation function themselves.

Implementation: The web application is written in Java and relies on a MySQL database for its storage needs. A request is initially emailed to a system designated email account which is monitored by a worker thread in the web application. These email requests are downloaded by the web application and saved into the database with a "pending" status. A given request then passes through a series of processing steps, as shown in figure 4. These steps use the AMT API



2011-03-22 11:57:52.0
2011-03-17 19:11:39.0
2011-03-17 12:56:04.0
2011-03-17 09:56:07.0
2011-03-16 06:32:05.0
2011-03-15 22:39:...

Figure 5: Mobile application. The left side, the *submit tab*, allows the user to take or choose an existing picture for submission. The right side, the *review tab*, allows the user to see the history of request-response pairs.

to handle submissions, polling for responses, and resubmitting answers for quality voting. When a final result becomes available, it is stored in the local database and returned to the user via email, as well as available from CrowdSight via a REST API.

3.2 Mobile Application

CrowdSight is intended to leverage the ubiquity of mobile devices with cameras. We provide multiple methods of interacting with services which range from email to custom mobile apps. The typical user of a CrowdSight service never needs to use the Crowdsight website, all interaction can be done through their mobile device.

Via email: CrowdSight is designed for quickly prototyping new services. To insure that it is easily available from any mobile device we provide an email interface for submitting requests and receiving results. An email request is composed of a service ID specified in the subject line and an attached photo. Results are returned via email. This interface was chosen as the lowest common denominator that would insure new services can be easily tested and prototyped in the field.

Sample Application: We provide a sample mobile application on the iOS platform. This phone application is simply a thin veneer on top of the functionality provided by the web application, which it accesses through a simple REST API. A screenshot of the sample phone app is provided in figure 5. This particular app simply allows a user to take a picture and submit it, as well as to review the results for all past requests.

Custom Application: Many developers will eventually want to wrap their service in a custom mobile application. This might be as simple as rebranding the sample application, recompiling it, and selling it in the mobile application marketplace. More advanced applications will want to *do* something with results. Because the interfaces to CrowdSight are extremely simple, and specifying new services is much easier than writing computer vision code to accomplish the same tasks, it is straightforward and easy to add CrowdSight functionality to more complex mobile applications.

4 Applications

In order to validate the suitability of CrowdSight as a platform for quickly prototyping new applications we let several graduate students create services they themselves would be interested in using. Following is a sample of those applications. They were able to go from concept to having a service up and running in at most a couple of hours.

Transcription: Identifying and transcribing text in photographs is an active area of computer vision research (Chen and Yuille 2004; Wu, Chen, and Yang 2005; Chen et al. 2010). Application domains include robotics, road sign detection, aids to the blind, and building book inventories. Unfortunately customized algorithms are typically needed for each new domain. We developed a transcription service which is independent of the scene clutter, font, handwriting, and domain by simply asking turkers to type in the text that they see. The output is a text file containing the text transcribed from the image. This application is so simple that we use it as the example template for service developers (development time: 10 minutes).

Counting people: Counting people in images is an important component of many real world systems. For example, Hyman built a system to count people in order to evaluate the efficacy of dynamic signs on increasing stair utilization (Hyman 2003). The end goal of the research reported in his thesis was the study on efficacy, but 50% of the project description was devoted to building the system itself. In order to get an estimate of time to build such a system, we asked the authors of a real time people counting algorithm how long it took to build their system (Yang, Gonzlez-baos, and Guibas 2003). They reported several man-months to develop and deploy the system. In contrast implementing a people counting algorithm took 30 minutes using the CrowdSight framework.

We tested our people counting application by monitoring a small conference room for a few hours, during which a meeting took place. A US$70 Panasonic PetCam was pointed at the room and configured to send an image via email to the CrowdSight service once per minute. We paid turkers US$0.01/image, and did not enable redundancy and voting. Figure 6 shows a challenging sample image sent to CrowdSight. In this case the service correctly identified that there are 5 people present. Existing computer vision methods would have trouble with this image, since one person is nearly completely obscured from view.

Figure 7 shows a plot of room occupancy over time. The gathered data accurately reflects reality: the room starts empty at 8:30AM, one meeting participant comes 40 minutes early, the remaining participants arrive a little late for the 10:00AM scheduled start, the meeting ends at 11:45AM, and by Noon the room is again empty. The single person at 8:50AM and 9:00AM is someone who entered the room briefly and then immediately walked out. During the meet-

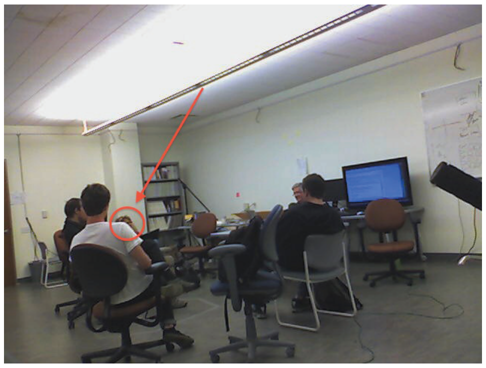


Figure 6: An example of a challenging image for automated people counting systems, which was correctly identified by turkers.

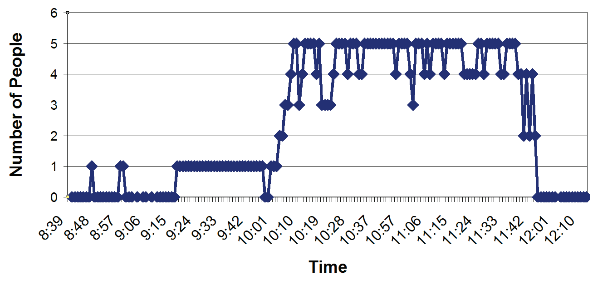
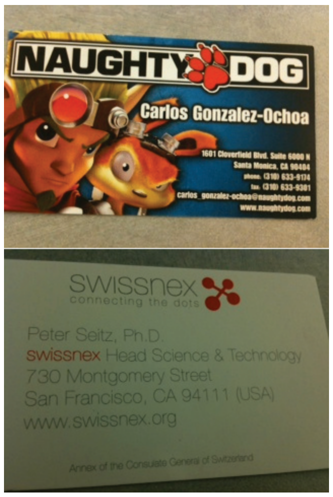


Figure 7: Plot showing room occupancy over a period of several hours as recorded by turkers. These numbers are fairly accurate (the variability is explained in part by people walking in and out during a meeting).

ing time, between 3 and 5 participants were visible to the camera. The service does make a few mistakes, missing some heavily occluded people during the meeting, however the overall data is surprisingly good.

Business card scanning: Many people would find useful taking a picture of a business card and giving it to an application which could magically transform it into an entry in their contacts database, so we created one such service in CrowdSight in about 15 minutes. This is a different problem than the freeform transcription described above since it requires the data be parsed into known fields (Saiga et al. 1993).

We created a structured form for turkers to enter data with 12 fields (company, first name, last name, street, city, state, zip code, phone number, mobile phone, fax number, email address, website address). This seems like the kind of application for which we should have good automated methods, however our tests of existing commercial applications show that this is not the case. Even though OCR is considered a solved problem in many respects, figuring out what information goes where is not necessarily trivial, even in a domain as limited as that of a business card. Figure 8 shows a couple of the business cards tested, drawn from a sample of 30 business cards, with field completion rates (on average how many fields were filled in correctly by the system per business card) of 80% for CrowdSight, 40% for BizSnap, and 30% for ABBYY's card reader (ABBYY).



NAUGHTY DOG
Carlos Gonzalez-Ochoa
1601 Cloverfield Blvd. Suite 6000 N
Santa Monica, CA 90404
phone. (310) 633-9174
fax. (310) 633-9301
carlos_gonzalez-ochoa@naughtydog.com
www.naughtydog.com

swissnex
connecting the dots
Peter Seitz, Ph.D.
swissnex Head Science & Technology
730 Montgomery Street
San Francisco, CA 94111 (USA)
www.swissnex.org
Annex of the Consulate General of Switzerland

Figure 8: Sample business cards tested by the CrowdSight service and the 2 automated apps. Neither of the automated apps recognized any of the fields correctly in the top card, and only 40% of the fields in the bottom card. The CrowdSight service was nearly 100% accurate on both, except for a misspelling in an email address.

For this service, we paid turkers US$0.05/card, whereas the rates of the commercial apps varied. ABBYY offered their card reader app for US$4.99 for processing an unlimited number of cards and BizSnap had a tiered pricing structure: US$1.00 for 10 cards, US$2.00 for 25, or US$6.00 for unlimited processing.

Product finder: This service enables a user to take a picture of something and to get back a link pointing to where the item in the image can be purchased. Object recognition is a well studied problem in computer vision, and shopping is just one possible application (Lowe 1999; Yeh et al. 2005; Tsai et al. 2008).

We compared the performance of this service against that of two existing applications: one which uses automated image processing algorithms, SnapTell, and one which requires the bar code of the item to be scanned, RedLaser. If an item had a bar code, RedLaser would usually find the item, however there were bar codes that we scanned that were not in RedLaser's database and also there are many items we tested which did not have a bar code. The SnapTell application, on the other hand, did not require bar codes but had a limited product category database (books, DVDs, CDs, and video games).

We measured the accuracy of the system tested by determining whether or not a valid link for purchasing the item was provided, which all systems are supposed to be able to do. In the end, all three systems had comparable accuracy, identifying most items (8 out of 10 on average) within their advertised domain; however, the CrowdSight service had a much more encompassing domain, not being restricted to items having a barcode nor to a limited product category.

This CrowdSight service was built in about 20 minutes.

5 Future Work

A possible future extension to the framework would be to enable developers to compose existing services into composite workflows. For example, defining a new service X to be the sequential composition of existing services A and B (any of which could itself be a composite service). We also envision improvements in the UI for creating services, since some developers would prefer a WYSIWYG interface. Our existing method of accuracy verification via rankings is extremely simple, and investigating more sophisticated methods that better control costs would be useful. Finally, it would also be desirable to be able to process media other than images, including audio and/or video input.

6 Conclusions

We developed a framework which simplifies prototyping applications that rely on "computer vision." The processing leverages human computation. Developers who use our framework are spared both from developing a complex computer vision algorithm, and from many of the details normally needed to deploy a solution using Amazon Mechanical Turk. End users of applications are never exposed to the implementation method, they just experience that the image gets processed and the results are accurate. We believe that the ability to quickly and cheaply prototype and deploy new applications will increase the rate at which useful applications are found. We expect that some of these solutions will continue to use a human computation back-end, while others will eventually migrate to a fully automated processing algorithm.

References

ABBYY. *ABBYY Business Card Reader*. http://www.abbyy.com/bcr˙iphone/.

AmazonRemembers. *Amazon Remembers*. http://www.amazon.com/.

AMT. *Amazon Mechanical Turk*. https://www.mturk.com/mturk/welcome.

Bernstein, M. S.; Little, G.; Miller, R. C.; Hartmann, B.; Ackerman, M. S.; Karger, D. R.; Crowell, D.; and Panovich, K. 2010. Soylent: a word processor with a crowd inside. In *ACM symposium on User interface software and technology*, UIST '10, 313–322. ACM.

Bigham, J. P.; Jayant, C.; Ji, H.; Little, G.; Miller, A.; Miller, R. C.; Miller, R.; Tatarowicz, A.; White, B.; White, S.; and Yeh, T. 2010. Vizwiz: nearly real-time answers to visual questions. In *ACM symposium on User interface software and technology*, UIST '10, 333–342. ACM.

BizSnap. *BizSnap*. http://www.bizsnap.cc/.

BusinessCardReader. *Business Card Reader*. http://www.shapeservices.com/.

Chen, X., and Yuille, A. 2004. Detecting and reading text in natural scenes. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. IEEE Computer Society Conference*, volume 2, II–366 – II–373 Vol.2.

Chen, D. M.; Tsai, S. S.; Girod, B.; Hsu, C.-H.; Kim, K.-H.; and Singh, J. P. 2010. Building book inventories using smartphones. In *International conference on Multimedia*, MM '10, 651–654. ACM.

Davis, J.; Arderiu, J.; Lin, H.; Nevins, Z.; Schuon, S.; Gallo, O.; and Yang, M.-H. 2010. The HPU. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference*, 9 –16.

Hyman, J. A. 2003. Computer vision based people tracking for motivating behavior in public spaces. *Thesis (M. Eng.) MIT, Dept. of Electrical Eng. and Comp. Sci.*

Little, G.; Chilton, L. B.; Goldman, M.; and Miller, R. C. 2009. Turkit: tools for iterative tasks on mechanical turk. In *ACM SIGKDD Workshop on Human Computation*, HCOMP '09, 29–30. ACM.

Lowe, D. G. 1999. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, ICCV '99, 1150–. IEEE Computer Society.

Parameswaran, A., and Polyzotis, N. 2011. Answering queries using humans, algorithms and databases. Technical report, Stanford InfoLab (TR-986).

Quinn, A., and Bederson, B. 2010. Human computation, charting the growth of a burgeoning field. Technical report, University of Maryland, Human-Computer Interaction Lab (HCIL-2010-26).

RedLaser. *RedLaser*. http://redlaser.com/.

Saiga, H.; Nakamura, Y.; Kitamura, Y.; and Morita, T. 1993. An OCR system for business cards. In *Document Analysis and Recognition, 1993., IEEE International Conference*, 802 –805.

SnapTell. *SnapTell*. http://www.snaptell.com/.

Snow, R.; O'Connor, B.; Jurafsky, D.; and Ng, A. Y. 2008. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *Empirical Methods in Natural Language Processing*, EMNLP '08, 254–263. Association for Computational Linguistics.

Tsai, S. S.; Chen, D.; Singh, J. P.; and Girod, B. 2008. Rate-efficient, real-time cd cover recognition on a camera-phone. In *ACM international conference on Multimedia*, MM '08, 1023–1024. ACM.

Wu, W.; Chen, X.; and Yang, J. 2005. Detection of text on road signs from video. *Intelligent Transportation Systems, IEEE Transactions on* 6(4):378 – 390.

Yan, T.; Marzilli, M.; Holmes, R.; Ganesan, D.; and Corner, M. 2009. mCrowd: a platform for mobile crowdsourcing. In *ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, 347–348. ACM.

Yang, D. B.; Gonzlez-baos, H. H.; and Guibas, L. J. 2003. Counting people in crowds with a real-time network of simple image sensors. In *IEEE International Conference on Computer Vision*, 122–129.

Yeh, T.; Grauman, K.; Tollmar, K.; and Darrell, T. 2005. A picture is worth a thousand keywords: image-based object search on a mobile platform. In *CHI '05 extended abstracts on Human factors in computing systems*, CHI EA '05, 2025–2028. ACM.