Early Implementation Experience with Wearable Cognitive Assistance Applications

Zhuo Chen, Lu Jiang, Wenlu Hu, Kiryong Ha, Brandon Amos,
Padmanabhan Pillai†, Alex Hauptmann, Mahadev Satyanarayanan

Carnegie Mellon University and †Intel Labs

ABSTRACT

A cognitive assistance application combines a wearable device such as Google Glass with cloudlet processing to provide step-by-step guidance on a complex task. In this paper, we focus on user assistance for narrow and well-defined tasks that require specialized knowledge and/or skills. We describe proof-of-concept implementations for four different tasks: assembling 2D Lego models, freehand sketching, playing ping-pong, and recommending context-relevant YouTube tutorials. We then reflect on the difficulties we faced in building these applications, and suggest future research that could simplify the creation of similar applications.

1. An Angel on Your Shoulder

GPS navigation systems have transformed our driving experience. They guide you step-by-step to your destination, offering you just-in-time voice guidance about upcoming actions that you need to take. If you make a mistake (e.g., miss an exit), this is promptly recognized and corrected. The difficult task of navigating an unfamiliar city has been transformed into a trivial exercise in following directions.

Imagine generalizing this metaphor. A *wearable cognitive assistance system* combines a device like Google Glass with cloud-based processing to guide you through a complex task. It feels just like following the directions of a GPS navigation system. You hear a synthesized voice telling you what to do next, and you see visual cues in the Glass display. When you make an error, the system catches it immediately and corrects you before the error cascades. This futuristic genre of applications is characterized as "astonishingly transformative" by the report of the 2013 NSF Workshop on Future Directions in Wireless Networking [1].

In its most general form, cognitive assistance is a very broad and ambitious concept that could be applied to virtually all facets of everyday life. Our initial goal is much more modest. We focus on user assistance for narrow and well-defined tasks that require specialized knowledge and/or skills. Figure 1 presents some hypothetical use cases.

In the past year, we have built proof-of-concept implementations for four different tasks: assembling 2D Lego models, freehand sketching, playing ping-pong, and recommending context-relevant YouTube tutorials. Although these are not yet production-quality implementations, they do address important real-world constraints and have taught us a lot. Our goal in writing this paper is to share the lessons and insights that we have gained from these implementations. We begin by briefly describing *Gabriel* [5], their common underlying platform. We then describe the four applications. Finally, we reflect on future research to simplify the creation of such cognitive assistance applications.

Sara sees Mr. Stanley drop his glass of water, clutch his chest and fall to the ground. She yells "Call 911" and heads for the wall-mounted Automated External Defibrillator. She dons the assistive Glass device that comes with it. It directs her to move Mr. Stanley away from the water. It then tells her to dry his chest before mounting the pads. When she tries to mount them symmetrically, it warns her that the left side is placed too high. Soon, the AED jolts Mr. Stanley back to life.

(a) Medical Training

Alice is a member of the Air National Guard. This weekend her assignment is to check the cockpit instruments on a cargo plane. She has observed the procedure just once, over a year ago. She asks her cognitive assistant for help. Her Glass device takes a picture of the instrument panel, recognizes the access point, and displays a picture of the panel with the access point highlighted. Alice is now easily able to take her readings, and finishes her task ahead of schedule.

(b) Industrial Troubleshooting

Figure 1: Cognitive Assistance Scenarios

2. The Gabriel Platform on OpenStack++

Cognitive assistance applications are both latency sensitive and resource intensive because they are shaped by the demands of human cognition. Many everyday tasks that humans perform effortlessly are exquisite feats of real-time analytics on multiple sensor stream inputs. For example, a human conversation involves many diverse inputs: the language content and deep semantics of the words, the tone in which they are spoken, the facial expressions and eye movements with which they are spoken, and the body language and gestures that accompany them. All of these distinct channels of information have to be processed and combined in real time. Similarly, an assistive system that augments cognition has to apply enormous computing resources to the same video and audio sensor streams to produce real time response to the human user.

For reasons that have been discussed extensively in earlier papers [5, 6, 15], the only viable approach to consistently meeting these processing and latency demands is to perform sensing and interaction on a wearable device (such

WearSys'15, May 18, 2015, Florence, Italy.

http://dx.doi.org/10.1145/2753509.2753517.

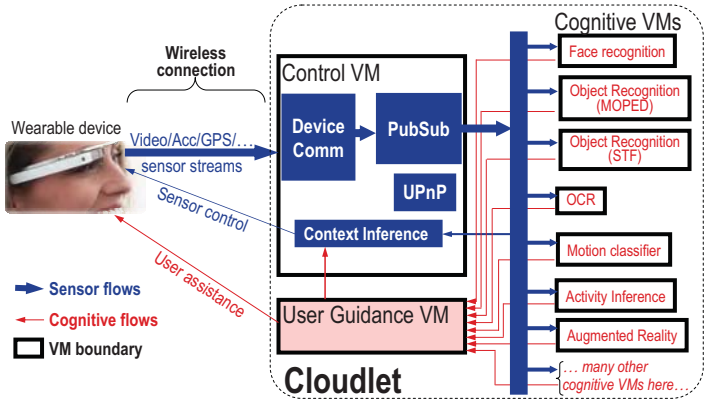


Figure 2: The Gabriel Architecture for Cognitive Assistance (Source: Ha et al [5])

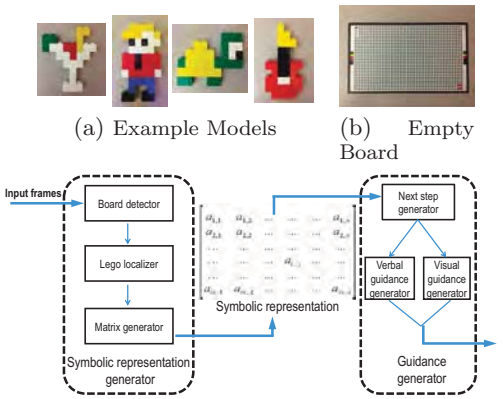


Figure 3: Lego Task

as Google Glass), while offloading almost all processing to a decentralized cloud computing element called a *cloudlet* that is located just one wireless hop away. Only through this approach can we offer crisp interaction while allowing devices to have acceptable battery life and to remain small, lightweight, and cool enough to wear.

Our cloudlets run Linux-based OpenStack++ [17]. This is a derivative of the widely-used OpenStack platform [14] for cloud computing, with extensions for cloudlet discovery, rapid provisioning, VM handoff, and other cloudlet-specific functionality. On top of OpenStack++, we have created the Gabriel platform for wearable cognitive assistance applications. Its goal is to allow developers to focus on application-specific issues, by factoring out difficult-to-implement functionality that is often needed by members of this class of challenging applications. Gabriel offers "plug-and-play" simplicity in creating or reusing *cognitive engines* such as face recognition, object recognition, and speech recognition.

Figure 2 illustrates Gabriel's back-end processing structure. An ensemble of cognitive engines, each encapsulated in a virtual machine (VM), independently processes the incoming flow of sensor data from a Glass device. In the applications described in Sections 3 to 6, the application-specific computer vision code is implemented as a cognitive VM. A single *control VM* is responsible for all interactions with the Glass device and preprocessing of incoming streams. Gabriel uses a publish-subscribe (PubSub) mechanism to distribute the sensor streams from control VM to cognitive VMs. The outputs of the cognitive VMs are sent to a single *User Guidance VM* that integrates these outputs and performs higher-level cognitive processing. From time to time, this processing triggers output for user assistance.

The *context inference* module in Gabriel understands a user's context using outputs from cognitive engines, and adjusts Glass's sensing and transmission policy accordingly. In its simplest form, it turns on/off a particular sensor based on applications' needs. For more fine-grained control, the *offload shaping* technique can be used [7]. For example, if all cognitive engines require sharp images as input, detecting blurry frames on the mobile device and dropping them before transmission can save wireless bandwidth, improve cloudlet scalability and enhance battery life. This *early discard* idea is a good complement to the general rule of preferential processing on cloudlets.

3. Lego Assistant

The first application we have implemented is a Lego Assistant that guides a user in assembling 2D models using the Lego product *Life of George* [13]. Figures 3(a) and 3(b) show some example Lego models and the board on which they are assembled. A YouTube demo can be found at `http://youtu.be/uy17Hz5xvmY`.

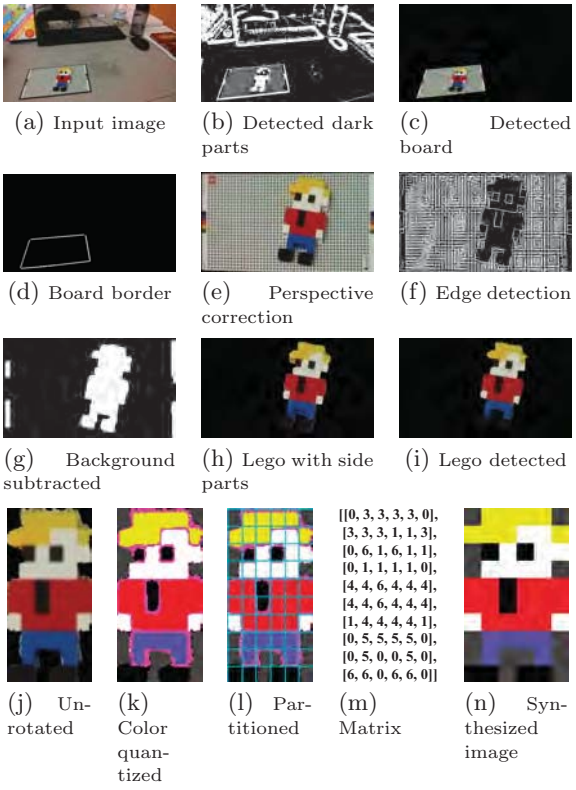
3.1 Cloudlet Workflow

Video from the Glass camera is streamed to the cloudlet, and processed there by the cognitive VM for this task. As Figure 3(c) shows, the processing workflow for each video frame has two major phases. In the first phase, the frame is analyzed to extract a *symbolic representation* of the current state of the Lego task. This phase has to be tolerant of considerable variation in lighting levels, light sources, position of the viewer with respect to the board, task-unrelated clutter in the image background, and so on. The symbolic representation is an idealized representation of the input image that excludes all irrelevant details. One can view this phase as a task-specific "analog-to-digital" conversion of sensor input — the enormous state space of the input image is simplified to the much smaller state space of the symbolic representation. Technically, of course, all processing is digital. As shown in Figure 4(m), the symbolic representation for this task is a two-dimensional matrix with values representing brick color. The second phase operates exclusively on the symbolic representation. By comparing it to expected task state, user guidance is generated.

This approach of first extracting a symbolic representation from the Glass sensor streams, and then using it exclusively in the rest of the workflow, is also how the other tasks (Sections 4 to 6) are implemented. We cautiously generalize from these four examples that this two-phase approach may be the canonical way to structure the implementation of any wearable cognitive assistance application. Broader validation will be needed to strengthen this observation.

3.2 Extracting the Symbolic Representation

We use the OpenCV image processing library in a series of steps that are briefly summarized below, using the video frame shown in Figure 4(a) as a working example. The first step is to find the board, using its distinctive black border and black dot pattern. Reliable detection of the board is harder than it may seem because of variation in lighting



(a) Input image (b) Detected dark parts (c) Detected board

(d) Board border (e) Perspective correction (f) Edge detection

(g) Background subtracted (h) Lego with side parts (i) Lego detected

(j) Unrotated (k) Color quantized (l) Partitioned (m) Matrix (n) Synthesized image

Figure 4: Lego: Symbolic Representation (Source: [16])

conditions. We subtract the original image by a blurred version of it, and then threshold the difference to give a robust black detector (Figure 4(b)). We find the boundary of the board (Figure 4(c)), then the four corners (Figure 4(d)), and then perform perspective transformation (Figure 4(e)).

Next, we extract the Lego model from the board. We perform edge detection (Figure 4(f)), then apply dilations and erosions to find the largest blob near the center of the board (Figure 4(g)). Unfortunately, sole reliance on edge detection is not robust. The sides of Lego bricks have more texture than the surface, leading to uncertainty in detection. We correct for this by finding the sides using color detection, adding them to the Lego shape obtained by edge detection (Figure 4(h)), and then performing erosion to remove the side parts (Figure 4(i)). The amount of erosion is calculated from the perspective transform matrix. For robust color detection, we use the grey world color normalization method [2].

In the final set of steps, we rotate the image to an upright orientation (Figure 4(j)). Each pixel is then quantized to one of the Lego brick colors (Figures 4(k) and 4(l)), with magenta color representing uncertainty. Final assignment of brick color is done by a weighted majority vote of colors within the block, with pixels near the block center being assigned more weight. Figure 4(m) shows the final matrix representation. Figure 4(n) is synthesized from this matrix.

3.3 Generating Guidance

A task in our system is represented as a linked list of Lego states, starting from the beginning state (nothing) to the target state (the user's goal). Based on the matrix representation of the current Lego state, our system tries to find an optimal next step towards the task target. If there is a clear next step, we convey this to the user using both verbal guidance (whispered instruction) and visual guidance (animation on the Glass screen). If the user fails to follow instructions, the user's current state may not be in the predefined state list. In such cases, our implementation tries to be smarter, as detailed in a recent paper [16].

(a) Target (b) Polygon (c) Feedback

Figure 5: Guidance by Drawing Assistant

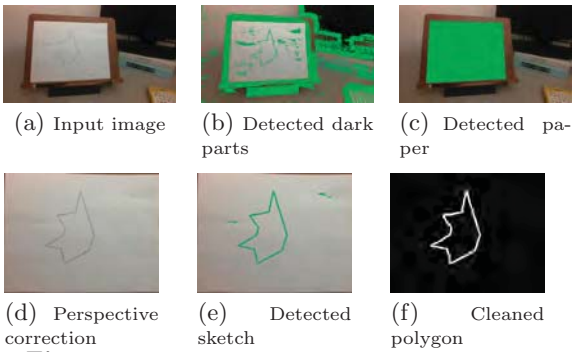
4. Drawing Assistant

In contrast to the Lego application, which is entirely new, our second application explores the challenges of modifying an existing application to provide wearable cognitive assistance. The *Drawing Assistant* by Iarussi et al. [8] guides a user in the classic technique of drawing-by-observation. As originally implemented, the application requires use of a special input drawing device, such as a pen-tablet. On the computer screen, it offers construction lines for the user to follow, recognizes the user's drawing progress, and offers corrective feedback. For example, Figure 5(a) shows a target drawing that the user is trying to copy. The blue polygon in Figure 5(b) is the outline generated by the Drawing Assistant, and Figure 5(c) shows the feedback provided by the application in response to the user's attempt to copy the polygon. The color of the construction lines in feedback images represents the correctness of different parts of the user's attempt: blue is good, red is bad. The dashed red lines indicate erroneous alignment between corners.

This application, as well as many other similar ones [12], require use of computer-friendly input (pen-tablet or mouse) and output (screen). We wondered whether we could use Google Glass to extend this application to work with arbitrary drawing instruments and surfaces: e.g., using pencil on paper, oil paint on canvas, or colored markers on a whiteboard. We have confirmed that this is indeed possible. Our approach is to use Google Glass and cloudlet processing to extract a symbolic representation of the user's input. We splice this extracted information into the existing guidance logic of the Drawing Assistant, and the rest of the system works with little change.

4.1 Extracting the Symbolic Representation

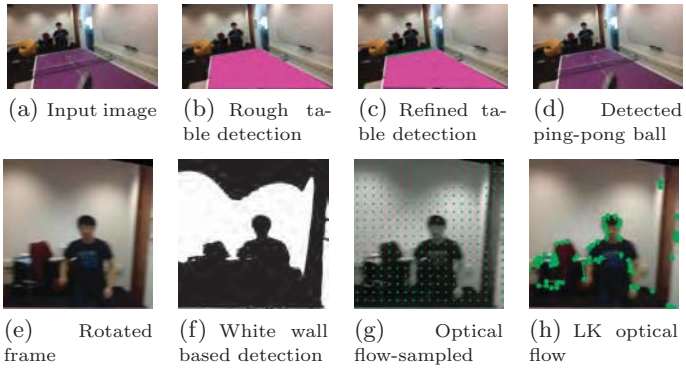
For ease of exposition, we assume that the user is drawing with a pen on paper. The processing for other media (e.g. colored markers on a whiteboard) is identical. To understand a user's drawing, the first step is to locate the paper from any arbitrary background. Similar to the black detection problem in the Lego assembly task, detecting "white" paper is not as easy as looking directly at the RGB values. Therefore, we use an approach similar to the board localization approach in the Lego Assistant to reliably detect the easel surface (Figure 6(b)). This is followed by a quadrilateral detection which represents the paper (Figure 6(c)).



(a) Input image (b) Detected dark parts (c) Detected paper

(d) Perspective correction (e) Detected sketch (f) Cleaned polygon

Figure 6: Drawing: Symbolic Representation



(a) Input image (b) Rough table detection (c) Refined table detection (d) Detected ping-pong ball

(e) Rotated frame (f) White wall based detection (g) Optical flow-sampled (h) LK optical flow

Figure 7: Ping-pong: Symbolic Representation

Line and corner detection are then applied to transform the paper image to a standard rectangle shape (Figure 6(d)).

The second step tries to identify all the pixels associated with the user's sketches. To do this, we use the difference of Gaussian approach and thresholding to find darker areas in the paper. However, as seen in Figure 6(e), this detection can be noisy because of shadows and creases. Therefore, we filter out the noisy parts based on their size, shape, and distance to other components. The result is a clean polygon (Figure 6(f)), which is the symbolic representation. As mentioned above, this symbolic representation is fed to the largely unmodified application in place of the pen-based input it was designed for, and the feedback is sent back to be shown on the Glass's screen.

5. Ping-pong Assistant

To explore the impact of tight real-time constraints, we have built a Ping-pong Assistant. Its goal is to help a user to choose the optimal direction to hit the ball to the opponent. Note that we are not trying to compensate for the lack of the user's own visual processing. That would be necessary, for example, if we were to tackle the far more challenging and difficult task of enabling a blind person to play. At this early stage of our work, our goal is merely to help a novice play a little better by whispering hints.

5.1 Extracting the Symbolic Representation

We use a 3-tuple as the symbolic representation. The first element is a boolean indicating whether the player is in a rally or not. The second element is a floating point number in the range from 0 to 1, describing the position of the opponent (extreme left is 0 and extreme right is 1). The third element uses the same notation to describe the position of ball. We generate this 3-tuple for each video frame.

Table detector: As shown in Figure 7(b), simple color detection can be used to approximately locate the table. To accurately detect the edges of the table in spite of lighting variation and occlusions by the opponent, we dilate the detected area, use white table edges to remove low possibility areas, and then use the Douglas-Peucker algorithm [4] to approximate the table area with polygons. Multiple iterations of this procedure yield a clean final result (Figure 7(c)). The top edge is also detected using the polygon information and marked in green in the figure.

Ball detector: For every area whose color is close to yellow, we determine if it is the ball based on its shape, size, position relative to table, and the ball's position in the previous frame. Once the ball is detected (Figure 7(d)), we find its coordinates in the input image and calculate its position relative to the table by using the transformation matrix described below for the opponent detector.

Opponent detector: Using the information obtained in table detection, we rotate and cut the area above the top edge of the table to get a square shape containing the opponent (Figure 7(e)). Within this standardized area, we use three approaches to locate the opponent. The first approach is simple and fast, but error-prone. It assumes that the background is mostly white wall, and selects the area with least white color (Figure 7(f)). The second approach is based on human motion calculated by optical flow, using two consecutive frames as input. It samples the image and calculates the flow at each sample point. The small green dots in Figure 7(g) are the sample points, while the arrows describe the direction and magnitude of flow. By appropriate low-pass filtering, we can find the position with strongest motion, and hence the opponent's likely location. The third approach is similar to the second, but calculates flow only at detected interest points (green circles in Figure 7(h)). The latency of the symbolic representation extractor is about 70 milliseconds when only the first approach is used, but increases by fifty percent when all the three approaches are combined using a weighted average to improve accuracy.

5.2 Generating Guidance

Guidance is based on a recent history of states. For example, if we see the opponent is standing at the right, and the ball has been hit to the right several times, we will suggest that the user hit to the left. Only speech guidance is provided, so the Glass screen is unused. The guidance is very simple, just "left" or "right." The same guidance will not be repeated within three seconds.

6. Assistance from Crowd-sourced Videos

Millions of crowd-sourced tutorial videos exist online, and YouTube itself hosts more than 83 million, covering a wide range of tasks. To leverage this valuable resource, we have built an application that uses video from Google Glass to characterize a user's context. It then searches a corpus of pre-indexed YouTube tutorial videos, and returns the best-match to be played on the user's Glass. Imagine a user learning to cook a new dish. Our application can deliver a YouTube tutorial on making this dish using similar tools. We call this application "YouTube Assistant" for brevity. It offers non-interactive tutorials for a wide range of user contexts by leveraging crowd-sourced YouTube videos.

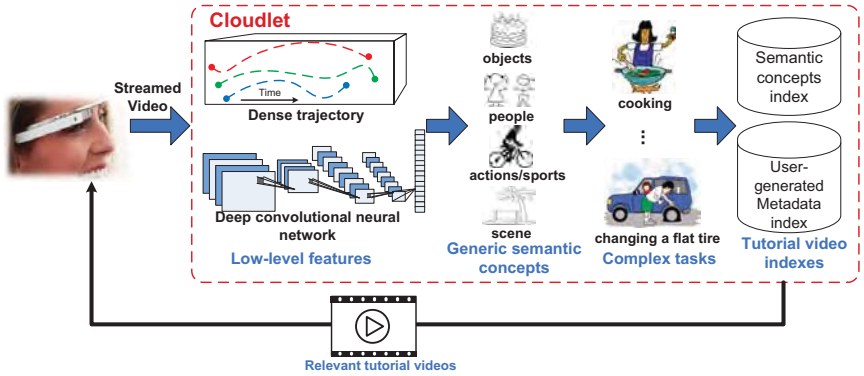


Figure 8: Pipeline for Discovering User Context

Semantic Category	Examples
People	male, baby, teenager, girl, 3 or more people
Scene	beach, urban scene, outdoor, forest
Object	car, table, dog, cat, guitar, computer screen
Action	shaking hand, cooking, sitting down, dancing
Sports	cycling, skiing, bullfighting, hiking, tennis

Figure 9: Concept Examples

6.1 Extracting the Symbolic Representation

In contrast to previous applications, we operate on short video segments rather than individual frames. Each segment is six seconds long, and two consecutive segments have a three seconds overlap. Multiple video segments are processed in parallel to improve throughput.

We compose previous works [9, 18] and use the workflow in Figure 8 to discover user context. We first extract low level features from the video segment, and then convert them into a list of generic semantic concepts, such as scene, object, person, and actions. This concept list is effectively the symbolic representation. Figure 9 lists some example concepts that we can detect. We further classify concepts into activity categories. The activity category is combined with the concept list to search for relevant tutorials.

Low level feature extraction: We extract dense trajectory feature descriptors [18, 19] from each video segment. This captures local points randomly sampled from each frame, and tracks the points in a dense optical flow field. Alternatively, low level features can be obtained from deep convolutional neural networks [11]. Searching directly based on the low-level features is extremely hard because they lack any semantic meanings and have very high dimensionality.

Concept detector: We apply our off-the-shelf concept detector trained by the SPCL pipeline [9] to extract semantic concepts. The 3,000+ semantic concepts recognized by this detector represent a broad range of real world events. The context detector is trained offline from over two million Internet amateur video segments [10].

Activity detector: We further characterize the user's task with higher level activities based on the concepts detected. For example, "cooking omelette" is an activity characterized by the presence of object "egg" and "butter", scene "indoor" and action "cooking" or "moving arm". By adding a concept layer before the activity detection, we significantly scale up the activities that can be detected.

6.2 Generating Guidance

We have downloaded about 72,000 tutorial videos from YouTube and indexed them based on the video's metadata (titles and descriptions) and the semantic concepts extracted from the video content. Once the user's concept list and activity is extracted, we use the standard language model [20] to find the most relevant video tutorial from our video pool. The YouTube link of the suggested video is then sent back to the user. If the user taps his Glass device, the tutorial will start playing using Android YouTube API.

7. Future Directions

When we began this work, the feasibility of providing step-by-step, closed-loop, task-specific guidance using a Glass-like wearable device was only a dream. We have now verified that such applications are indeed feasible. The Lego, Drawing, and Ping-pong Assistants all react to a user's task-relevant actions with helpful guidance. The YouTube Assistant is a hybrid: it uses sensor inputs to infer context, and then uses this inference to provide guidance via a YouTube video. While many improvements are possible in these Assistants, the feasibility of wearable cognitive assistance is no longer in doubt. The versatility, generality and ease of use of Gabriel as a "plug-and-play" platform is also confirmed by our experience. Balancing these positive observations, we have identified three areas where substantial improvement is needed. We discuss these below.

7.1 Faster Prototyping

Much of the difficulty in building a cognitive assistant lies in the computer vision part. While robust recognition from diverse viewpoints and lighting conditions is a well-understood problem for computer vision experts, it is still a challenge for non-experts. OpenCV helps, but is not enough. Even for a job as simple as color detection, we still have to spend many hours in tuning the parameters and making it robust across different lighting conditions. An easy-to-use, GUI-based toolkit for quick testing and debugging of OpenCV-based functions could significantly improve the application development cycle and enhance productivity.

Applications should also be able to easily reuse state-of-art computer vision algorithms. For example, many assembly tasks may depend on the best object detection library. Facilitating this requires modification to the existing Gabriel architecture. We can add another layer of *library VMs* between the control VM and cognitive VMs. They provide results shared by many cognitive VMs, and can be easily upgraded as new libraries are available. Tools to rapidly adapt an existing detector to a specific application would also help. This is currently a slow process involving manual curation of training data and building each model afresh.

Implementing task-specific guidance is easy when user state is limited, but becomes much harder as the state space expands. In the Lego Assistant, if we allow a user to select his own path to building a model, the current approach of

matching a user's state to a pre-defined list no longer works. In such cases, the concept of "guidance-by-example" may help. For example, could we record multiple experts performing a task, extract the correct sequence of state changes using symbolic representation extractor, and then suggest an optimal next step based on these sequences?

7.2 Improving Runtime Performance

Extraction of a symbolic representation using computer vision tends to be slow, even on a cloudlet. For example, the YouTube Assistant currently takes one minute to extract context from a six-second video. Parallelism can offer significant speedup in computer vision. Processing multiple frames in parallel is easy, but exploiting parallelism within a single frame is harder. We need tools such as Sprout [3] to simplify this effort.

Generally, there is a tradeoff between the accuracy and speed of a computer vision algorithm. The different opponent detection approaches in the Ping-pong Assistant is a good example. Combining different algorithms for higher accuracy has been extensively studied in the computer vision community. We wonder if we can combine different approaches for speed as well. For example, the accuracy of an algorithm usually depends on image content, lighting conditions and background. Since these are unlikely to change much during a single task execution, we could test different algorithms in parallel at the start of the task and then select the optimal one for the rest of the task.

7.3 Extending Battery Life

The battery life of Glass is only tens of minutes for the applications described here. In addition, it tends to get too hot to be comfortable. There is a clear need to exploit task-specific opportunities for energy efficiency. For example, there is typically a region of interest (ROI) that captures task-related information in an image. The board in the Lego Assistant and the paper in the Drawing Assistant are examples. The ROI typically does not move much between consecutive frames, and this can be used to reduce data transmission and cloudlet processing.

We can also exploit the fact that full processing and new guidance are only needed when task state changes. It may be possible to perform cheap and early detection on the Glass device that verifies that task state has *not* changed. Using the concept of offload shaping [7], those frames can be dropped before transmission from Glass.

Acknowledgements

We wish to thank Bobby Klatzky and Dan Siewiorek for the scenarios in Figure 1(a) and Figure 1(b) respectively. Rahul Sukthankar, Jan Harkes, and Benjamin Gilbert helped us in designing the computer vision processing described in Section 3. This research was supported by the National Science Foundation (NSF) under grant numbers IIS-1065336 and IIS-1251187. Additional support was provided by the Intel Corporation, Google, Vodafone, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and should not be attributed to their employers or funding sources.

8. REFERENCES

[1] S. Banerjee and D. O. Wu. Final report from the NSF Workshop on Future Directions in Wireless Networking. National Science Foundation, November 2013.

[2] J. M. Buenaposada and B. Luis. Variations of Grey World for face tracking. In *Image Processing & Communications 7*, 2001.

[3] M.-Y. Chen, L. Mummert, P. Pillai, A. Hauptmann, and R. Sukthankar. Exploiting multi-level parallelism for low-latency activity recognition in streaming video. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, 2010.

[4] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2), 1973.

[5] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan. Towards Wearable Cognitive Assistance. In *Proceedings of the Twelfth International Conference on Mobile Systems, Applications, and Services*, Bretton Woods, NH, June 2014.

[6] K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies, and M. Satyanarayanan. The Impact of Mobile Multimedia Applications on Data Center Consolidation. In *Proceedings of the IEEE International Conference on Cloud Engineering*, San Francisco, CA, March 2013.

[7] W. Hu, B. Amos, Z. Chen, K. Ha, W. Richter, P. Pillai, B. Gilbert, J. Harkes, and M. Satyanarayanan. The Case for Offload Shaping. In *Proceedings of HotMobile 2015*, Santa Fe, NM, February 2015.

[8] E. Iarussi, A. Bousseau, and T. Tsandilas. The drawing assistant: Automated drawing guidance and feedback from photographs. In *ACM Symposium on User Interface Software and Technology (UIST)*, 2013.

[9] L. Jiang, D. Meng, Q. Zhao, S. Shan, and A. G. Hauptmann. Self-paced curriculum learning. In *AAAI*, 2015.

[10] L. Jiang, S.-I. Yu, D. Meng, T. Mitamura, and A. G. Hauptmann. Bridging the ultimate semantic gap: A semantic search engine for internet videos. In *ACM International Conference on Multimedia Retrieval*, 2015.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] Y. J. Lee, C. L. Zitnick, and M. F. Cohen. Shadowdraw: real-time user guidance for freehand drawing. *ACM Transactions on Graphics (TOG)*, 30(4), 2011.

[13] Lego. Life of George. `http://george.lego.com/`, October 2011.

[14] OpenStack. `http://www.openstack.org/`, February 2015.

[15] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4), October-December 2009.

[16] M. Satyanarayanan, Z. Chen, K. Ha, W. Hu, W. Richter, and P. Pillai. Cloudlets: at the Leading Edge of Mobile-Cloud Convergence. In *Proceedings of 6th International Conference on Mobile Computing, Applications and Services (MobiCASE)*, Austin, Texas, USA, November 2014.

[17] M. Satyanarayanan, R. Schuster, M. Ebling, G. Fettweis, H. Flinck, K. Joshi, and K. Sabnani. An Open Ecosystem for Mobile-Cloud Convergence. *IEEE Communications Magazine*, (3), March 2015.

[18] H. Wang and C. Schmid. Action recognition with improved trajectories. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 3551–3558. IEEE, 2013.

[19] S.-I. Yu, L. Jiang, Z. Mao, X. Chang, X. Du, C. Gan, Z. Lan, Z. Xu, X. Li, Y. Cai, et al. Informedia@ TRECVID 2014 MED and MER. In *NIST TRECVID Video Retrieval Evaluation Workshop*, 2014.

[20] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342. ACM, 2001.