Early Implementation Experience with Wearable Cognitive Assistance Applications

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

April 2015
CMU-CS-15-103

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[†]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.

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.

Keywords: cognitive assistance, wearable computing, mobile computing, Google Glass, cloudlet, cloud offloading, offload shaping, computer vision

1 Introduction

Instructors are extremely helpful in human's learning experience. They teach you how to perform a complex task by offering verbal guidance and demonstrating the procedure. When you make progress, they congratulate you and provide instructions for the next step. If you make an error, they quickly catch it and tell you how to fix it.

In the past year, we have been developing a *wearable cognitive assistance* system that acts as a *virtual instructor* to offer similar help. It combines a wearable device such as Google Glass and cloud processing to guide you through a complex task. It uses wearable sensors such as camera and microphone to continuously capture a user's progress, and leverage modern computer vision and signal processing techniques to understand the user's state. Verbal feedback are then whispered into the user's ears, along with visual cues to be displayed on Glass's screen. Figure 1 presents some hypothetical use cases.

In its most general form, a cognitive assistant could offer guidance to virtually all facets of everyday life, running twenty four by seven. For example, a recent work [5] targets improving social life quality for people under cognitive decline. However, the lack of robustness in computer vision algorithms, the network instability as a user moves, and the limitation in battery life of wearable devices, all make this dream extremely challenging. Therefore, our initial goal focuses on user assistance for narrow and well-defined tasks within a short period of time. We use task assistance to refer to this special genre of cognitive assistance applications.

In this paper, we provide detailed description of four task assistance applications that we have recently built, as an extended version of a recent paper [3]. The first application offers step-by-step instructions to help a user assemble two dimensional Lego models. The second modifies existing software to teach drawing-by-observation techniques. The third application helps a user to play better Ping-pong. Finally, the fourth application tries to help with multiple tasks. It delivers relevant YouTube tutorial videos to a user based on the user's context.

These implementations have confirmed the feasibility of building such task assistance applications, and have taught us a lot. Our goal in writing this paper is to share the lessons we have learned, and to identify valuable research directions. In the next section, we briefly review Gabriel [5], the common platform under all applications. From Section 3 to Section 6, we describe details of each application one by one. Finally, we discuss challenges in building such applications and possible solutions in Section 7.

2 The Gabriel Platform on Cloudlet

A cognitive assistance application augments human perception by offering visual and verbal cues. This is very latency sensitive as it is shaped by the demands of human cognition. It also requires intensive computation to apply multiple complex algorithms to the incoming video and audio streams to extract meaningful feedback. For reasons that have been extensively discussed in previous works [5, 6, 14], the only viable approach to meet both the latency and compute requirement is to use a wearable device such as Google Glass to collect sensor data and offer feedback, while offloading almost all the computation to a cloudlet. A cloudlet is a small distributed cluster that is one wireless hop away from a wearable device, offering low-latency, high-bandwidth communication and rich computational resources. Only through this approach can we offer crisp

Jane wants to try butterscotch pudding today for the first time. She starts her "Recipe" application in Glass, selects butterscotch from pudding menu. After making several steps correct, the Glass whispers "Good job! Now gradually whisk in one cup of cream and stir it until smooth". Jane poured half a cup, since this is all she has at home, hoping this is not a critical step. This is quickly caught by the Glass, which then yells "This is not enough. Please go and buy more."

(a) Cooking

Bob just moved to Pittsburgh and bought a lot of new furniture from Ikea. Different from past experience of reading from instruction sheets, he now starts the "Assembly Assistant" Glass application to guide him in the assembly steps. Bob feels it's much easier to read instructions from Glass and finishes everything ahead of schedule. When he tries to sit on the chair he last assembled, the Glass warns him that he didn't screw the last nail tight enough...

(b) Furniture Assembly

Figure 1: Task Assistance Scenarios

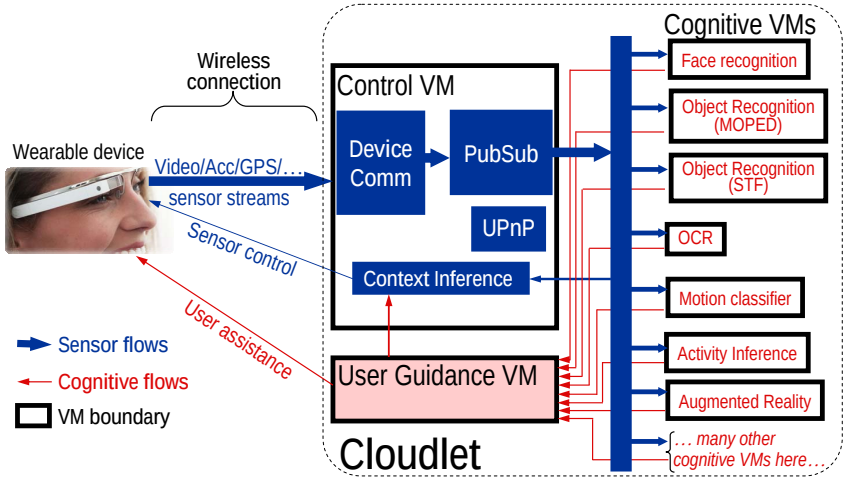
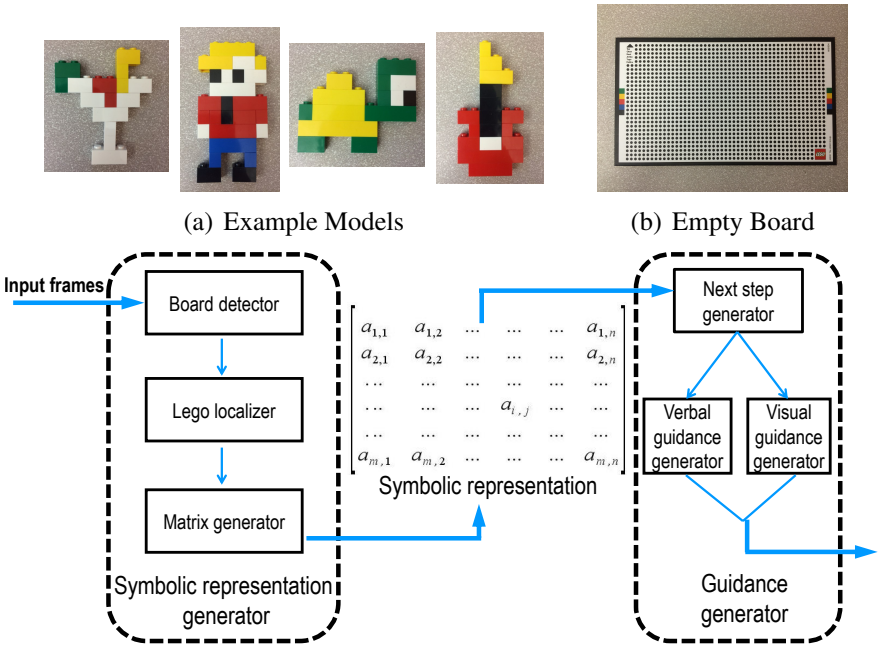


Figure 2: The Gabriel Architecture for Cognitive Assistance (Source: Ha et al [5])

interaction while allowing devices to have acceptable battery life and to remain small, lightweight, and cool enough to wear.

We have created the Gabriel [5] platform that runs on top of a cloudlet for cognitive assistance applications. Its goal is to provide common system and library support for most of the applications, allowing developers to focus on application-specific challenges. Particularly, Gabriel offers "plug-and-play" simplicity in reusing existing cognitive applications and developing new applications. Moreover, it uses application-layer flow control mechanism to guarantee low latency communication between a wearable device and different application software.

Figure 2 illustrates Gabriel's back-end processing structure. An ensemble of cognitive applications, each encapsulated in a virtual machine (VM) to form a *cognitive VM*, independently processes the incoming flow of sensor data from a Glass device. A single *control VM* is responsible for all interactions with the Glass device. It also does common preprocessing, such as down sampling or reformatting of incoming streams, to serve all cognitive VMs. 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.



(a) Example Models (b) Empty Board

(c) Workflow

Figure 3: Lego Task

The *context inference* module in Gabriel understands a user's context using outputs from cognitive VMs, 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 applications 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.

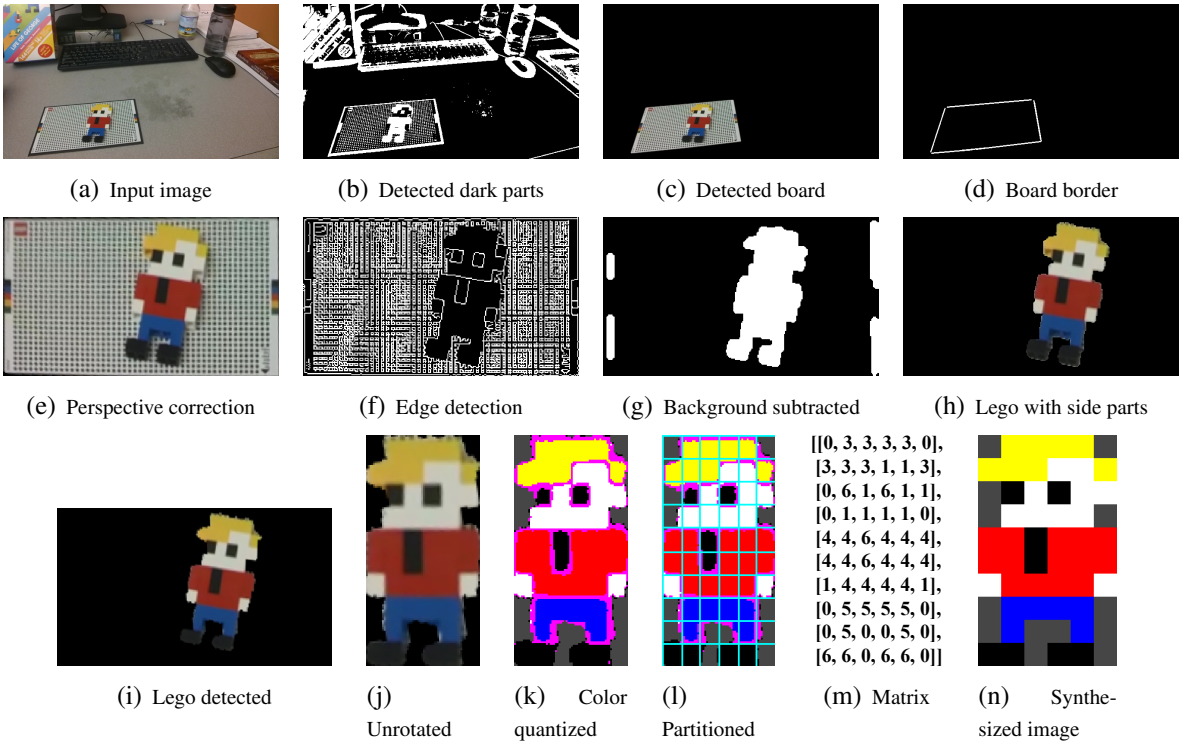
All of the following task assistance applications run on top of Gabriel. Each of them is implemented as a cognitive VM and receives a user's video streams from the PubSub system.

3 Lego Assistant

The first application we have implemented is a Lego Assistant that guides a user in assembling two dimensional Lego models using the Lego product *Life of George* [13]. Figures 3(a) shows some example models a user can build. From step to step, the Lego models are put on the board (Figure 3(b)) that comes with the product to be recognized. A YouTube video demonstrating this application can be found at `http://youtu.be/uy17Hz5xvmY`.

3.1 Cloudlet Workflow

Video captured by Glass camera is streamed to the cloudlet. 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



(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: [15])

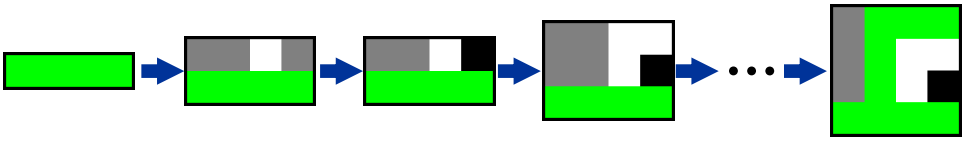
board, task-unrelated clutter in the image background, and so on. The symbolic representation is an idealized representation of the input image that excludes all irrelevant details. One can view this phase as a task-specific "analog-to-digital" conversion of sensor input — the enormous state space of 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. Figure 4(a) is an example input frame. As Figure 3(c) shows, the symbolic representation generator for each frame consists of three steps: board detector, Lego localizer, and matrix generator.

Board detector: The first step is to find the board, using its distinctive black color and dot pattern. Reliable detection of black color is harder than it may seem. Sole reliance on thresholding on the absolute brightness would fail because of variation in lighting conditions. Therefore, we subtract the original image by a blurred version of it, and then threshold the difference to give a robust



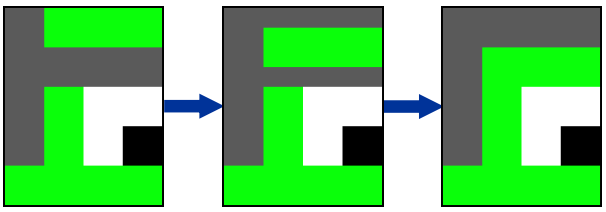
(a) An example of state list representation of task

"As a first step, please find a 1x4 green piece and put it on the board."

"Great! Now find a 1x2 red piece and put it on the top left of current model. "

"You are quite close. Now slightly move the 1x1 yellow piece to the right by 2 brick size."

(b) Example of verbal guidance messages



(c) Example of visual guidance animation

Figure 5: Example of guidance generation for Lego Assistant (Source: [15])

black detector (Figure 4(b)). This effectively leverages the relative brightness value rather than the absolute one.

A rectangular shape near the dense black dots are considered to be the board (Figure 4(c)) We then extract the boundary and four corners of the board by doing line detection and intersection calculation. Perspective transformation is performed to convert the board to a standard view (Figure 4(e)).

Lego localizer: We perform edge detection to distinguish the Lego model from the board background (Figure 4(f)). We 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, especially the heavy colored ones (e.g. yellow, red, etc.) 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 gray world assumption [1] to adjust the colors on board before detecting them.

Matrix generator: 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 a synthesized image representation for this matrix.

3.3 Generating Guidance

A Lego task is represented as a linked list of Lego states, starting from the beginning state (nothing) to the target state (the user's goal). Figure 5(a) shows an example of our task. Based on the matrix representation of the current Lego state, our system tries to find an optimal next step towards the task target. If the user's state falls into the state list of task representation, then we simply convey

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

Figure 6: Guidance by Drawing Assistant

information about the next state in the list. If a user fails to follow the instruction, an naive approach is to tell the user to revert back to the previous state. Our system tries to be smarter than this, using the technique detailed in [15].

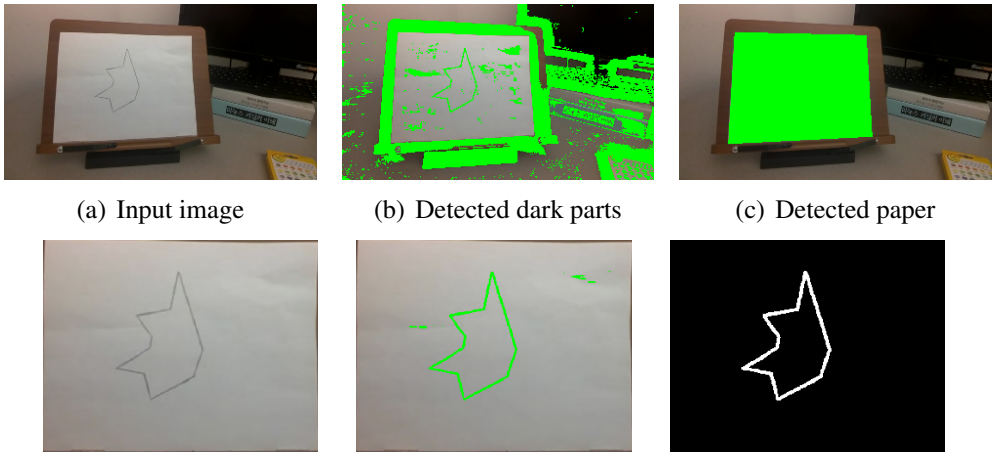
The guidance delivered to a user consists of both verbal and visual parts. The verbal guidance is whispered to the user through Android text-to-speech API, describing the next brick to pick and where it should be placed. The visual guidance displays an animation on Glass's screen, showing the same process as explained in the verbal instruction. Figure 5(b) and (c) show some guidance examples.

3.4 Performance

The Lego Assistant has been demonstrated live at different occasions and has shown very robust performance. The extraction of symbolic representation takes about half a second for full processing of an image, and much less if some of the steps are not needed. For example, if no clear board can be detected in an image, Lego localizer and matrix generator won't execute. We also require two consecutive frames to generate the same state result before delivering guidance to the user. This ensures high accuracy, but doubles the response time as a user makes progress.

4 Drawing Assistant

In contrast to the Lego Assistant, which is entirely new, the second application explores the challenges of modifying an existing application to provide wearable cognitive assistance. Our Drawing Assistant is built upon previous work by Iarussi et al. [8] that guides a user in practicing classic drawing-by-observation technique. As originally implemented, the application automatically generates outline for a target object and displays it on a computer screen for the user to follow. It then recognizes the user's sketches on a pen-tablet and offers feedback information on the computer screen. For example, Figure 6(a) is a target drawing that a user tries to copy, and the blue polygon in Figure 6(b) is the outline generated by the Drawing Assistant. Figure 6(c) shows an example feedback image provided by the application in response to the user's attempt to copy the polygon.



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

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

Figure 7: Drawing: Symbolic Representation

This application, as well as many other similar ones [12], requires the drawing to be done on a computer-friendly input device, such as a pen-tablet. Our Drawing Assistant uses 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. It first extracts drawing-relevant information from an arbitrary image and converts a user's sketches into a clean binary image. This image serves as the symbolic representation of this application, and is essentially what the original software gets directly from the pen-tablet input. We splice this extracted information into the existing guidance logic of the application, and the rest of the system works with little change.

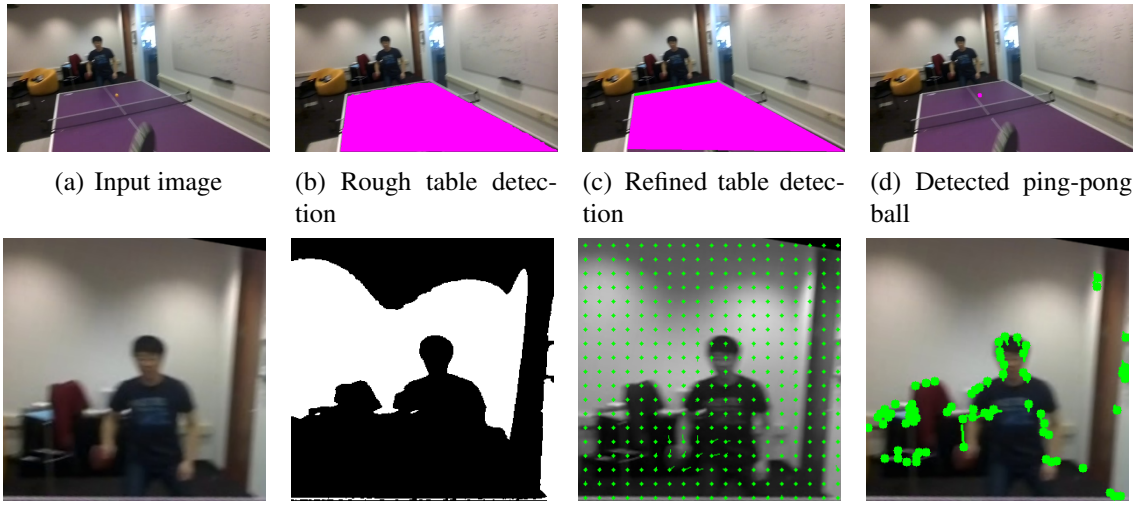
4.1 Extracting the Symbolic Representation

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 similar. 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. It thresholds on the difference between the input image and its blurred version, which results in a reliable detection of the easel surface (Figure 7(b)). This is followed by a quadrilateral detection which represents the paper (Figure 7(c)). Line and corner detection are then applied to perspectively transform the paper image to a standard rectangle shape (Figure 7(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 7(e), this detection can be noisy because of shadows and creases on the paper. Therefore, we filter out the noisy parts based on their size, shape, and distance to other components. The result is a clean polygon (Figure 7(f)), which is the symbolic representation.

4.2 Generating Guidance

The symbolic representation mentioned above is fed to the largely unmodified application in place of the pen-based input it was designed for. The application uses Shape Context descriptor to com-



(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 8: Ping-pong: Symbolic Representation

pare the polygon shape drawn by user and that shown in the target. It then sends back the feedback image to be shown on the Glass's screen. For example, Figure 6(c) is an example feedback derived from a user's drawing in Figure 7(f) and target outline in Figure 6(b). Note that the color of the outline polygon in feedback images represents the correctness of different parts of the user's attempt: blue is good, while red means not good. The dashed red lines indicate erroneous alignment between corners.

4.3 Performance

The symbolic representation extraction for Drawing Assistant works reasonably accurate, and has a latency of tens of millisecond. We have tested the system with several users. While some of them complain about the limited size of Google Glass display for recognizing detailed feedback information, they all appreciate the idea of capturing drawings on different media using Google Glass to offer feedback. They also wish the application to be extended to provide feedback for general sketches, but not only the outline polygons.

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, where the suggested direction is based on the observed ball and opponent position. 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 have a better chance to win by whispering hints.

5.1 Extracting the Symbolic Representation

In our prototype, 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 pair of consecutive video frames.

Table detector: As shown in Figure 8(b), simple color detection can be used to approximately locate the table. However, this is not robust because of lighting variations and sometimes occlusions from the opponent. For example, the table surface close to the top table edge is not included in the rough detection. To fix this, 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 8(c)). The top edge is also detected using the polygon information and marked in green in the figure.

Opponent detector: To detect the opponent, we focus on the area above the top edge of the table. Using the table information obtained above, we rotate and cut the area above the top edge of the table to get a square shape containing the opponent (Figure 8(e)). Within this standardized area, we use three approaches to locate the opponent. We combine the three approaches using weighted average to achieve higher accuracy.

The first approach assumes that the background is mostly white wall, and selects the area with least white color (Figure 8(f)). This approach is simple and fast, but error-prone when the background is not clean. 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 8(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. Note that when the Glass wearer moves, the background objects will also move and confuse the motion detector. To compensate for this, the strongest motion area is selected after subtracting all optical-flow vectors by the average of these vectors. The third approach is similar to the second, but calculates flow only at detected interest points (green circles in Figure 8(h)).

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 8(d)), we find its coordinates in the input image and calculate its position relative to the table by using the transformation matrix described above for the opponent detector.

5.2 Generating Guidance

Guidance is generated 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. The Ping-pong assistant will offer speech guidance only, since the user's eyes will be focusing on tracking the ball. The guidance is very simple, just "left" or "right." The same guidance will not be repeated within three seconds.

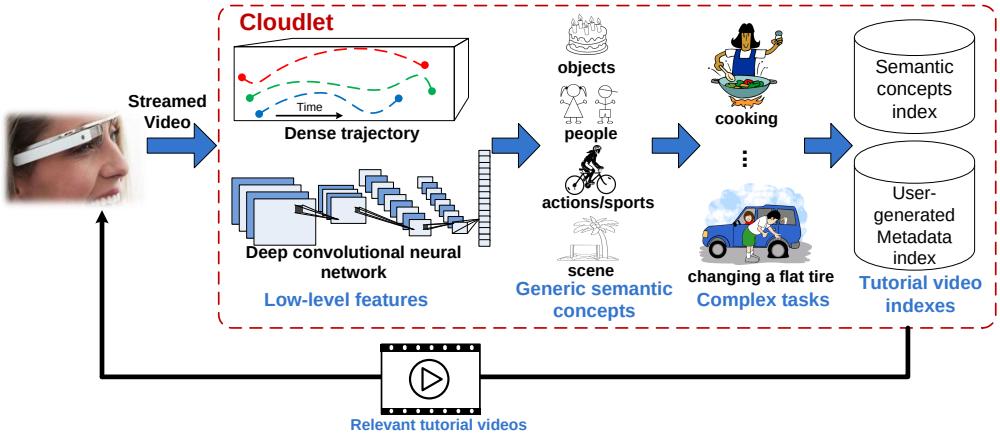


Figure 9: Pipeline for Discovering User Context

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 10: Concept Examples

5.3 Performance

As a first step, we have collected a small number of videos capturing the viewpoint of one user playing ping-pong in a certain playing environment. The parameters for the symbolic representation extractor are tuned on these videos and the algorithm works reasonably well for this dataset. The accuracy degrades as we apply the algorithm to different users because of variations in users' habits in playing ping-pong and their different viewpoints. In terms of speed, the latency to extract symbolic representation is about 70 milliseconds when only the white-wall approach for opponent detection is used, but increases by fifty percent when all the three approaches are used in combination to improve accuracy.

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. We wondered whether we could leverage them to provide assistance for different tasks. To explore this possibility, we build an application that can 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 instructing to make the same dish using similar tools. We call this application YouTube Assistant for brevity. It offers useful, though non-interactive, tutorials in a wide range of user contexts by leveraging the crowd-sourced YouTube videos.

6.1 Extracting the Symbolic Representation

In contrast to previous applications, YouTube assistant operates on short video segments rather than individual frames. Therefore, the first step on the cloudlet is to group image streams to short video files. In our prototype, each video segment is six seconds long, and two consecutive segments have a three seconds overlap. This overlap yields a higher chance to capture user's actions. Multiple video segments are processed in parallel to improve throughput.

We compose previous works [9, 16] and use the workflow in Figure 9 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 10 lists some example concepts that we can detect. We further classify concepts into activity categories. The activity category, combined with concept list, is then used to search for relevant tutorials.

Low level feature extraction: We extract dense trajectory feature descriptors [16, 17] from each video segment. This captures the 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 dimension.

Concept detector: We apply our off-the-shelf concept detector from SPCL pipeline [9] to extract semantic concepts. The detector is trained offline from over two million Internet amateur video segments [10]. Training on these videos reduces the domain difference between the training data and the testing video captured by Glass. The 3,000+ semantic concepts recognized by this detector represent a broad range of real world events.

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 have significantly scaled up the activities that can be detected.

6.2 Generating Guidance

We have downloaded around 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 [18] 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.

6.3 Performance

The context detector in this application consumes large amount of time. On a modern desktop workstation with Intel® Core™ i7 4-core processor and 32GB of memory, it takes about one minute to extract concepts from a six second video segment. We have spawned multiple processes to run the context detector in parallel, which has significantly improved system throughput. However,

improving latency is much harder. Adding more computing resources won't help due to the single-thread nature as the code is written. We will have to rewrite the algorithm in a more parallel way to reduce system response time.

7 Future Directions

In this work, we implement proof-of-concept cognitive assistance systems to provide step-by-step, closed-loop, task-specific guidance using a Glass-like wearable device. The Lego, Drawing, and Ping-pong Assistants all react to a user's task-relevant actions with predefined guidance. The YouTube Assistant is a little different: it first recognizes a user's context and then delivers guidance through searching crowed-sourced tutorials at runtime. The versatility, generality and ease of use of Gabriel as a "plug-and-play" platform is confirmed by our implementation experience. While the feasibility of wearable cognitive assistance is no longer in doubt, we have also identified areas where substantial improvement is needed. Below we discuss these improvement possibilities from three aspects.

7.1 Faster prototyping

Developing a new task assistant is complex and challenging today, especially to make it robust in different environments. We wonder if some higher-level libraries and toolkits could simplify the process of prototyping a new application.

Better interface for OpenCV: Much of the difficulty lies in using computer vision technologies to extract the symbolic representation. We have used OpenCV extensively ourselves. It 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 make it robust across different lighting conditions. We believe an easy-to-use, GUI-based toolkit for quick testing and debugging of OpenCV-based functions can significantly improve the application development cycle and enhance productivity.

Easy to share library support: Applications should also be able to easily reuse state-of-art computer vision algorithms. For example, a lot of assembly tasks may all depend on the most accurate object detection library, or the context detection technique as used in YouTube Assistant. Facilitating this requires some modification to 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.

Generating guidance for complex tasks: Generating guidance is simple when user state space is small. A developer can list all expected user states and their corresponding guidance. The Lego Assistant is one example. As long as a user follows the instructions in every step, appropriate guidance can be made by matching the user's state to the pre-defined list in the database. Things become much more complex as user state space expands: if we allow a user to select his own path to build a Lego model, it's impossible to know user states ahead of time. In such cases, the concept of "guidance-by-example" may help. For example, could we record multiple experienced users performing a task, extract the correct sequence of state changes using symbolic representation

extractor, and then suggest an optimal next step by matching a new user's state to these sequences? The suggestion quality will increase as more users use the application.

7.2 Improving runtime performance

Extraction of a symbolic representation using computer vision tends to be slow, even on a cloudlet. For example, as mentioned in Section 6, the latency for YouTube Assistant is more than one minute. Parallelism can usually offer significant speedup in computer vision algorithms. This involves exploiting both inter-frame parallelism to improve system throughput, as well as intra-frame parallelism to reduce latency. Processing multiple frames in parallel can be achieved by spawning different VMs for the same task. Exploiting parallelism within a frame is much harder. It usually needs significant restructuring of original code, such as making the code run on a GPU. We need tools such as Sprout [2] 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 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.

It is noticeable that not all cognitive assistance applications have the same latency requirement. For example, the Ping-pong Assistant requires tens of millisecond response time, while the Drawing Assistant can probably tolerate sub-second latency. The Gabriel system should provide a way to easily identify an application's needs, and adjust resource allocation accordingly.

7.3 Extending battery life

The battery life of Glass is only tens of minutes for the applications running on top of Gabriel. In addition, it tends to get too hot to be comfortable. We should note here that the Gabriel client is not doing any heavy computation. It merely transmits images and delivers visual and verbal guidance to a user.

There is a clear need to exploit task-specific opportunities to reduce the transmission traffic for energy efficiency. For example, there is typically a region of interest (ROI) that captures task-related information in an image. The board in Lego Assistant and the paper in Drawing Assistant are examples. Since the ROI typically does not move much between consecutive frames, we can use the concept of offload shaping 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 continuous cheap processing, such as observing accelerometer values, on the Glass device to see if a user's state has changed. Only when it changes do we need to capture frames and transmit them. Moreover, if we have a reasonable expectation of how long a user needs to complete each step, we could temporarily stop all processing on Glass at the beginning of a step, and resume after some expected time.

References

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

[2] 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.

[3] Z. Chen, L. Jiang, W. Hu, K. Ha, B. Amos, P. Pillai, A. Hauptmann, and M. Satyanarayanan. Early Implementation Experience with Wearable Cognitive Assistance Applications. In *Proceedings of the 1st ACM Workshop on Wearable Systems and Applications (WearSys)*, Florence, Italy, 2015.

[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] 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.

[15] 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.

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

[17] 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.

[18] 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.