A-Eye – leveraging ML models for helping low-vision and blind people

Torekhan Shynar 20160882 Lee GeonJu 20170445 Woo SeungYeon 20170414 Ahmadli Mirali 20170847

The project is under the **Option 1: Apply the existing models or techniques to your own problem**. **Repository of the project** is available here: `https://github.com/Shynar88/A-Eye`

I. Introduction

Researchers make tremendous efforts to enable machines to see and comprehend the world around us. Numerous state-of-the-art models have impressive performance and accuracy. They are mostly deployed in the areas such as autonomous vehicles, security, surveillance and robotics. However, those models also have tremendous potential for facilitating people with visual impairments. Our team leverages multiple models for helping blind and low-vision people though the interface of mobile application. As the result of the project, we were able to develop fully functional application which satisfies functional and non-functional requirements specified, and were able to make contribution to the original model by fixing several faults and releasing scripts for the automatic set up of environment for all the models used. We provide application with the service oriented architecture which can be scaled up to encompass multiple models.

A. Motivation

Our team was driven to solve real life problem, which can help people and positively impact their lives. Total population of planet Earth: 7.7 billion as of October 2019 according to the most recent United Nations estimates. According to World Health Organization(WHO): “Globally, at least 2.2 billion people have a vision impairment or blindness”, which contributes to 28.6% of people on the Earth. Moreover, there are 285 million blind and low-vision people, which is 5.6 times the population of Republic of Korea. Thus, there is tremendous number of people whose lives could be positively affected by AI models.[1]

B. Problem setting

During problem setting we made research to analyze the needs and requirements of the target audience. We conducted requirements elicitation process as well as market analysis in order to identify existing methods used by target audience in order to find breakpoints and problems which should be addressed. This stage had significant influence on the design of our system.

C. Market research and Related Work

We analyzed currently available solutions in order to understand breakpoints people have while using them. There are various technologies released, however, each of them has some kind of limitation.

1. Application provides limited functionality. Because of this, user needs to install multiple applications where each of them corresponds to one functionality. For example, "LookTell" provides only money identification functionality.

2. Unusable UI. The GUI of numerous applications is hard to use even for people with good vision(ex: numerous buttons, complex navigation, too small buttons to press). For example, Microsoft's "Seeing AI" has 13 GUI elements located in one screen, which makes it hard to navigate even for people with perfect vision.

3. Human factor. There are crowd-sourcing applications like "Be My Eyes", which connect people with visual impairments and volunteers who are willing to help. However, application doesn't have defence system against people who mock blind people by intentionally telling wrong information. Moreover, the demand exceeds the supply: there are more people seeking for help, than people willing to help.

4. High cost and potential health hazard. There are startups like OrCam, Aipoly which provide system integrated with smart glasses though which user can be connected with the agent from the call center. However, there are expenses on smart glasses and payment for the specially hired agents. In addition, constantly wearing electronic device on the head has potential health hazards.

D. Requirements elicitation

The first choice of Requirements Elicitation method was interviewing and ethnographic observation. However, we couldn't find enough low-vision and blind people. Thus, we used another method of requirements elicitation - Storyboarding. Storyboarding is one of the main techniques along with interviewing utilized in industry for the requirements elicitation purposes. Three scenarios were storyboarded to reflect possible situations were application can be utilized.
In the figure 2 the situation where user needs to distinguish objects is depicted. In the figure 3 the situation where user needs to read the text from object is depicted. In the figure 4 the situation where user needs to get information on surrounding environment is depicted. From the storyboarding process and analysis, we identified following Functional and Non-functional requirements presented in the following subsection.

E. Functional requirements

No.	Description
FR1	User should get verbal information on the objects detected in front of the camera by tapping once on the screen
FR2	User should get verbal information on the description of scene in front of the camera by swapping up on the screen
FR3	User should get verbal information on the text detected in front of the camera by swapping down on the screen

TABLE I. Functional Requirements

F. Non-functional requirements

No.	Description
NFR1	The control over application should happen by gestures
NFR2	There should be no GUI elements
NFR3	Feedback should be given to the user immediately once the process started
NFR4	The completion time for the task shouldn't exceed 30 seconds(response time)
NFR5	The user should be able to accomplish needed task by using minimum number of actions: maximum 1 gesture should be required

TABLE II. Non-functional requirements

II. Method

A. System Architecture and Design

By analyzing Functional and Non-functional requirements, it was clear that for providing verbal information TTS model should be used, for the object detection - object detection models, for text detection - OCR model, and for surrounding scene description - image captioning model need to be utilized. From the analysis of requirements it was concluded that the application will have three main functionalities described in the functional requirements. We decided to use client-server model in order to have lightweight client and scalable server. The overall system architecture is presented on the figure 1. The Service Oriented Architecture(SOA) was used. In this way, each model on the server is completely modular - having own setup, installation and inference codes. In this way, if the load and demand for the certain model will be high, the number of container instances running that model can be up-scaled(replicated) on the server. In addition, if certain model needs to be removed, or new model needs to be added, this can be done easily, because models do not have any connection between each other and are called only by the coming request from the client. On the client side the Swift programming language is used. It is high performance oriented language for the native mobile development. MVVC (Model-View-ViewController) pattern was used. The client renders the camera input to the screen and has gesture recognizer listeners. When certain gesture recognizer is triggered, the photo is taken, compressed for faster network transmission, the TCP connection is established with the server and POST request is done which uploads image to the server. The server identifies the model which needs to be used by looking on the POST request and triggers the inference function of requested model and returns the results to the client. Upon successful response from the server, the client conveys the results to the user in the verbal form. For the object detection pretrained Detectron2[2] model from Facebook research, for the OCR - Tesseract[3], for the Object Detection "Show and Tell"[4] models are used. For the TTS, initially SV2TTS[5] model was used. However, it had very slow inference time and there were some noises. Thus, we switched to the Siri TTS[6] from Apple Research. It is highly parallelised and has no noises. In addition, it is highly compatible with the client side. We still provide the setup and inference codes of SV2TTS on the server, thus the developer can choose between two TTS models.

III. Additional Contributions

1. When setting up the models to run inference we realized how challenging it is to setup the server environment properly and download all the requirements, because some authors don't specify the list of dependencies utilized or the specific versions of the dependencies used. Moreover, when combining several models together, the dependencies conflicts can occur. After resolving all dependencies and environment issues, we wrote bash and python scripts which install all needed requirements with the specific version needed(discovered after days of debugging) and setup the server environment. All the setup ans installation codes for models are available in the GitHub repository provided.

2. Some of the models provided inference codes. However, they were not optimal. For example, each time when inference function is called, the tensorflow graph is built or it did extra work which was not needed for our goals(getting bounding boxes and etc) or for the TTS, the voice recording was needed. We optimized the inference codes so that one-time operations happen in the installation stage and during the inference, the code only directly related to the inference and our goals is called. In case of voice recording, we utilize prerecorded voice.

3. While writing inference code, we made several bug fixes in the "Show and Tell" model for image captioning. The first bug was - there were mismatches of some variables' names in Tensorflow graph. It was fixed writing modify_files() function. Secondly,

with the original code from git repository, it was impossible to get right result, because of bug in list indexing. Fixed by modifying line 182 in caption_generator.py. Thirdly, the original git repository says that Tensorflow ¿= 1.0.0 can be used, but actually, various errors can occur depending on the version of Tensorflow. As solution, the Tensorflow 1.0.0 should be used to avoid errors.

IV. Contributions

- Torekhan Shynar: team leader and project manager. Developed the mobile application(client side). Integrated Siri TTS with the application. Developed the server. Set up communication between client and server. Made requirements elicitation(storyboards) and market research. Wrote report. Draw System Architecture.
- Lee GeonJu: wrote set up, installation and inference scripts for the Image Captioning model. Made bug fixes in the Image Captioning Model released. Contributed to the Challenges and Discussions section.
- Woo SeungYeon: wrote set up, installation and inference scripts for the Object Detection and TTS models. Contributed to the Challenges and Discussions section. Contributed with ideas on the project evaluation and use-case scenarios for demonstration.
- Ahmadli Mirali: set up of Tesseract OCR. Assembled videos together for demonstration. Researched on accuracy of the models used in the project.

V. Evaluation and Results

The demonstration of the application: `https://www.youtube.com/watch?v=arOweCoOOTk&feature=youtu.be`. The application doesn't have the limitations found in current alternatives: no costs, no need for purchasing additional hardware(only smartphone is needed), take out human out of the loop, has user friendly UI for blind people(control fully happens through gestures), has several models integrated into one application, has lightweight(memory footprint and speed) client app. All the Functional and Non-Functional Requirements are satisfied. The accuracy of the application corresponds to the accuracy of the pretrained models utilized.[2][3][4][5][6]

VI. Discussion and Conclusions

We realized that setting up the needed environment with all requirements for the models is challenging task, because the documentation on dependencies is absent or slightly documented, and because of the dependencies conflicts between different models. It was Our results can be further improved by adding more hardware to the server(using stronger computational power), parallelising some code in order to increase the speed. More powerful models with higher accuracy can be utilized. The project can be extended by adding more models, thus more functionalities. As the result of the project, we were able to develop fully functional application which satisfies functional and nonfunctional requirements specified, and were able to make contribution to the original model by fixing several bugs and releasing scripts for the automatic set up of environment and dependencies for all the models used.

[1] WHO statistics
[2] Detectron2
[3] Tesseract
[4] Show and Tell
[5] SV2TTS
[6] Siri TTS

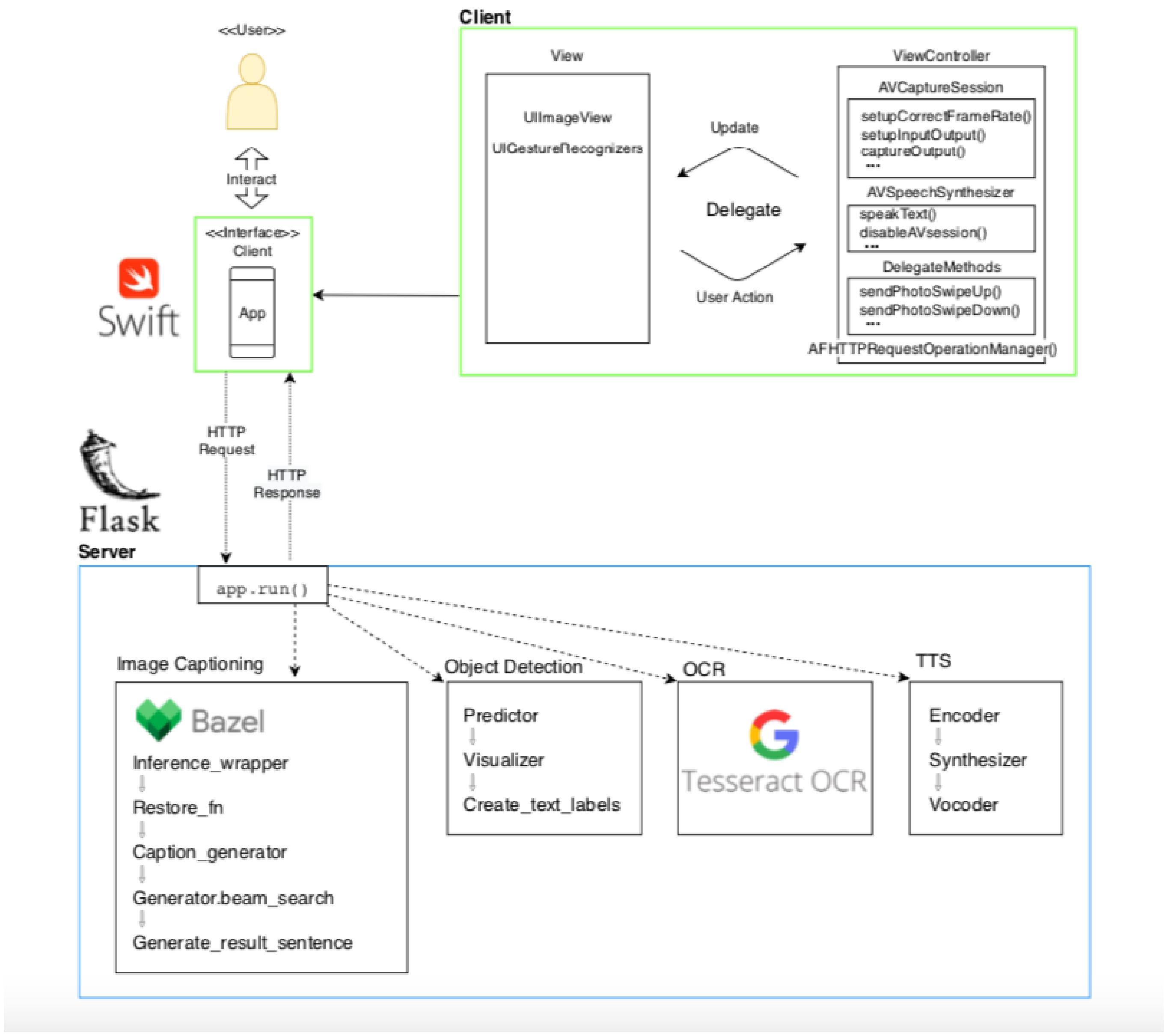

FIG. 1. System Architecture

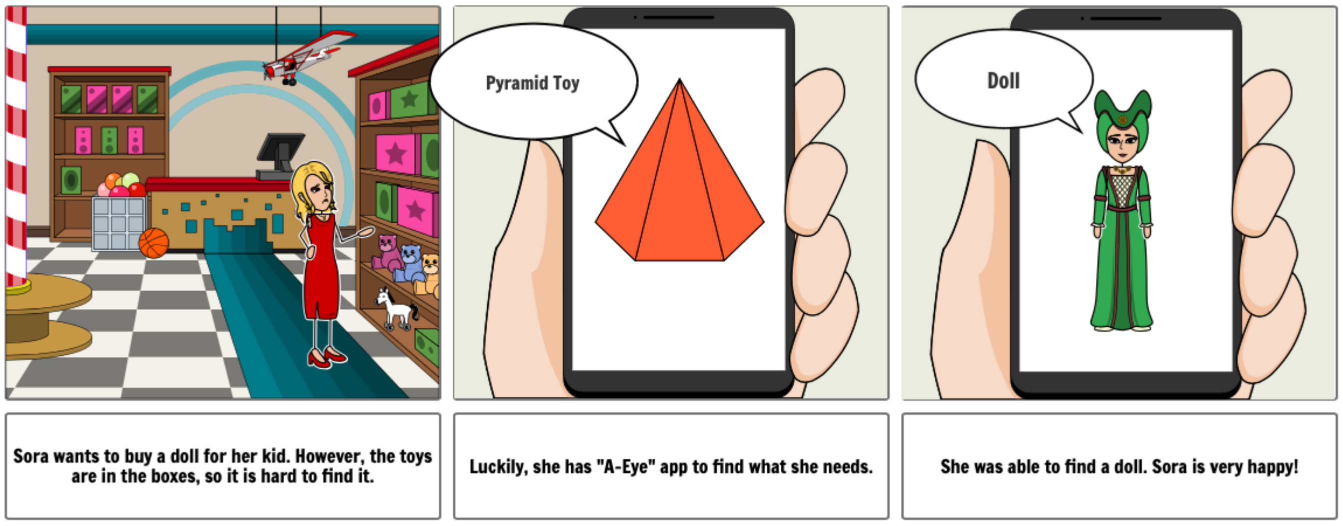
Pyramid Toy
Doll
Sora wants to buy a doll for her kid. However, the toys are in the boxes, so it is hard to find it.
Luckily, she has "A-Eye" app to find what she needs.
She was able to find a doll. Sora is very happy!

FIG. 2. Object Detection Storyboard

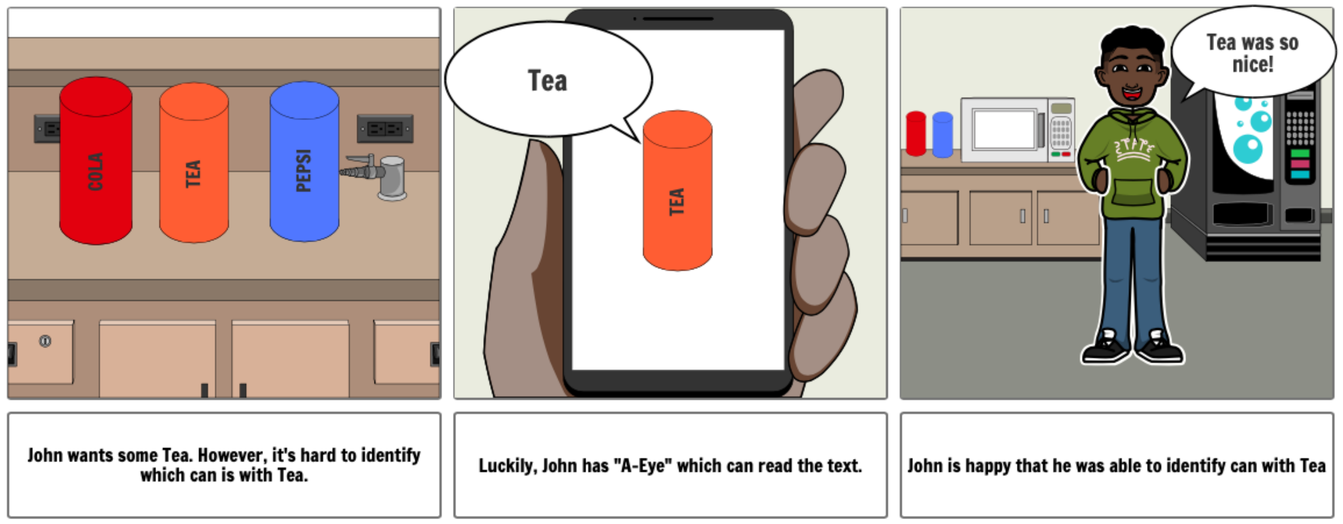
COLA
TEA
PEPSI
Tea
TEA
Tea was so nice!
John wants some Tea. However, it's hard to identify which can is with Tea.
Luckily, John has "A-Eye" which can read the text.
John is happy that he was able to identify can with Tea

FIG. 3. Text Detection Storyboard

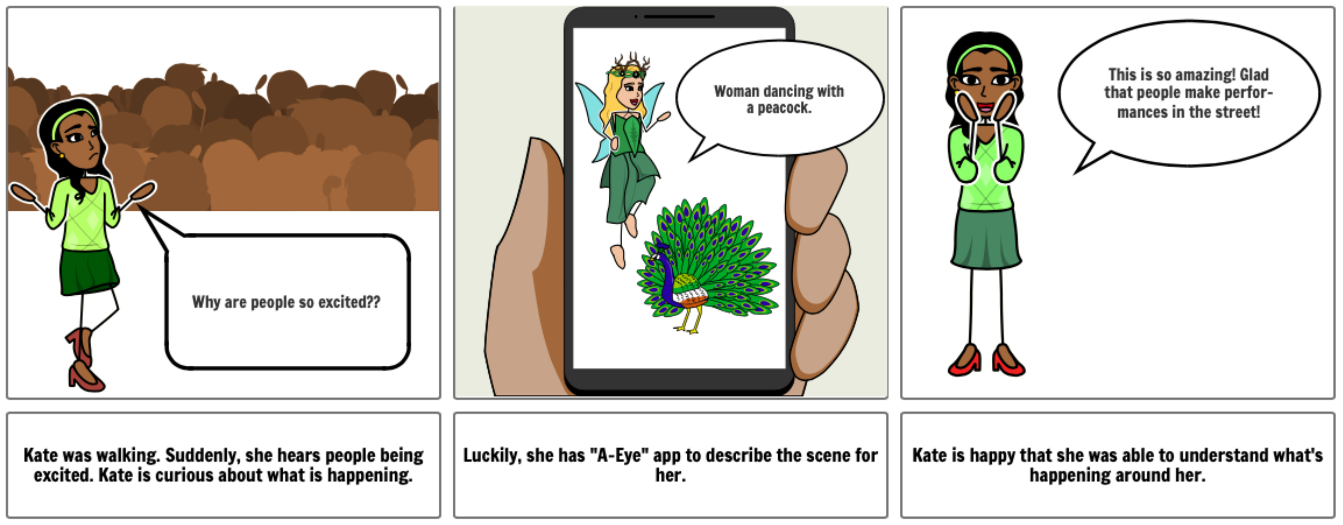
Why are people so excited??
Woman dancing with a peacock.
This is so amazing! Glad that people make performances in the street!
Kate was walking. Suddenly, she hears people being excited. Kate is curious about what is happening.
Luckily, she has "A-Eye" app to describe the scene for her.
Kate is happy that she was able to understand what's happening around her.

FIG. 4. Scene Description Storyboard