Spring 4-15-2020

SIGHT HELPER

Misael Uzcategui

SIGHT HELPER

by

Misael Uzcategui ID:231463

Applied Project report submitted to the Faculty of the Graduate School of the

HARRISBURG UNIVERSITY OF SCIENCE AND TECHNOLOGY

in fulfillment of the requirements for the degree of
Master of Science in Computer and Information Sciences

Supervised by: Abrar Qureshi, Ph.D.
Spring 2020

Table of Contents

Introduction to the Problem

There are significant numbers of people diagnosed with blindness or with low vision problems. In 2004 the Eye Diseases Prevalence Research Group estimated that more the (3%) of Americans the age of 40 or older are considered legally blind. The consideration for legally blind is for people that have less of 20 degrees of vision or a visual acuity up to 20/200 and visually impaired as 20/40.[2] The visual disability represents a daily struggle in their lives; any modification on the environment disrupts the mental map of their surroundings. Usually, these miss orientations avoided with guidance from a helper the guide through the changes in the settings, or in the worst-case scenario, the person with a visual disability will stumble with the new changes in the environment and reshape the mental map.

With the understanding of the amount of people that constantly struggle with the constant shifting environments. The need to of solutions that can provide independences to the people visual impaired is a constant request. However, is important to recognize the great effort that multiple companies and individuals have done to mitigate the problem, unfortunately the actual solutions are far from perfect and most of them are too expensive for the common user.

Previous research to Sight Helper

Many individuals, companies and universities have made efforts to provide solutions to people that are visual impaired, creating a foundation to future developments. In the case of Sight Helper use a significant amount of research as part of the foundations for the self-contain and mobile version.

Previous Research for Self-contain Version

In the area of object recognition and implementation of the Raspberry Pi have being done a significant amount of research that have guide in the process of development of Sight Helper. Bellow will be a small description of the research and the connection with Sight Helper.

The Electronics and Communications Engineers Abaya, Basa, Abad and, Dadios in 2014 did an investigation where they implement the computer vision library OpenCV with the self-contain board Raspberry Pi as security system. This investigation provides an important reference of the uses, integration and implementation of the algorithms of OpenCV into the Raspberry Pi.[1]

The Engineers Bhanse and Jaybhaye in 2018 work in a process for face detection using Raspberry Pi board. They describe the impact of the reduction of resolution in the number of frames for the time processing.[2]

Cameron and Islam both Electronics and Computer Engineers in 2018 work in an implement the consider the hue, saturation, and value with computer vision to do object detection in real-time systems.[3]

The Electrical Engineer Chang in 2014 work in the implementation of a three-layer neural network using a reference insect brain structure to create an agent for object recognition.[4]

The researchers Guennouni and Mansouri in 2014 work into the implementation of a cascade classifier using computer vision to do object detection in real-time systems.[5]

The Engineers Kaymak and Ucar in 2018 work into the evaluation and the implementation of multiple algorithms to establish recognition of objects and set instructions to a robotic arm through a Rasberry Pi.[6]

The researchers Mazumdar, Sarasvathi and, Kumar in 2017 work in the implementation into training of a convolutional neural network and a fully connected neural network by supervising learning to object recognition in a video by sequential frame extraction.[7]

The Computer Engineers Salih and Omer in 2018 presented an implementation of the Raspberry Pi as a video server with the protocols HTTP and RTP.[9]

The researchers Shah and Kapdi in 2017 presented a several alternatives to the application of deep neural network-based in convolutional networks in the process of object detection. [10]

The researchers Tepelea, Buciu,, Grava, Gavrilut, and, Gacsadi in 2019 presented an application of computer vision to recognize traffic signs using Raspberry Pi as a platform. The authors experiment with the resolution and scale factor to evaluate the detection distance and the time of processing.

Previous Research for Mobile version

In the efforts to provide solutions to the problem of not having sight or losing the vision capabilities there are many apps and devices that provide significant help to users. A few of those solutions are:

The authors- provides an implementation of a Neural Network with Euclidean Distance to create a model that looks for the maximum amount of light and detect the most relevant color. Also the system provide a framework that allow to expand the training of the model with new objects to recognize. At the end of any recognition the system provides a feedback through the speech synthesizer of the device.

The authors under the support of the University of Rochester and Microsoft Research did and incredible work presenting the challenges the blind people go through every day. The research presents the results of a year of data collection from the mobile app VizWiz which provide feedback from the community to user based on a question and a picture that the user takes.

The authors under the Design Information & Thinking Lab from Taiwan and the Innovative User Interface Lab from Taiwan develop the app BlindNavi which purpose is to provide a help in the navigation of blind people. The app provides a voice feedback of possible sensory clues.

The authors present a well description of the interactions mobile devices and problems the blind people experience using them, doing an special emphasis on smartphones. In the research process the authors develop a calendar that allow interaction with voice commands. As conclusion the authors present some lessons learned in the process like what must be an appropriate feedback and the suitable aesthetics that the application should have.

The authors present an evaluation of serverless computing in the cloud platform. The case of study is the AWS Lambda where the authors go through the complete process deploying, monitoring and evaluation of limitations. Concluding with the results of the tool GammaRay for tracking and evaluation of the serverless applications.

The authors present a navigation application that provide a voice feedback to the vision-impaired person. The application uses the floor plan as part of the initial base and with the Wi-Fi signal strength calculate the location. For the path calculation the authors implemented the Dijkstra's Algorithm and for the navigation through the application an implementation of the Google Voice Recognition service.

The authors provide an interesting benchmark between two of the main recognition services available in the market based on cloud service, Google Cloud Vision and AWS Rekognition. The article used both services to evaluate the readings of electricity and gas meter. The authors in order to evaluate the services define a set of images where they modify the scale, blur, gamma and noise. The results were close to each other. However, AWS Rekognition provide a 3% higher accuracy.

The authors provide an interesting database with the purpose to test object recognition algorithms with controlled settings. The database CURE-OR (Challenging Unreal and Real Environments for Object Recognition) provide a set of 1,000,000 images as a representation of 100 objects with variations in their orientation, color, texture and size. In the article the authors evaluate the performance of AWS Rekognition and Microsoft Azure Computer Vision.

The authors provide a cost comparation between the three most common architecture implemented for web applications, AWS Lambda that is a serverless architecture that is activated on http request, Monolithic where a preset of instances working nonstop host the application, and Microservices where the application is divided in microservices that are hosted in flexible instances that can upgrade and downgrade on demand. The article presents a significant reduction in cost and similar average response time across the architectures.

The authors present an application that implement Visual Search Engine in mobile devices to provide feedback to blind people. The authors present a methodology of collecting multiple frames and classify the quality of the frames and only send the key frames to the search engine. The fundamental argument is that base on VizWiz more than 17% of the questions are not answered because of the quality of the photo.

Sight Helper Purpose

Sight Helper look forward to engaging the emerging computer vision technology to provide an affordable solution for blind people or with low vision. The present project involves two different solutions to cover special purposes. A self-contain version that main purpose is to provide a self-contain device an affordable price that provide audio feedback of the objects using machine learning without the necessary of internet connection. Also, a mobile version that take advantage of the AWS Rekognition service as the machine learning engine behind the object recognition process.

Sight Helper Scope

Sight Helper as general scope is to improve the life of blind people or low vision provide guidance through the use of machine vision in a low cost. It covers tow possible solutions, the self-contain version that take the advantage of the powerful and simple Raspberry Pi as the platform for the computer vision model and peripherical. The other alternative is the mobile version that take advantage of the cloud computer vision service of AWS Rekogniton. In both solutions the simplicity is a key feature where the interaction happens by one button the trigger the photo and the recognition process.

Definitions, Acronyms, and Abbreviations

Definition / Acronyms	Description
Framework	A set of libraries to provide support to an specific task
API	Serie of protocols or functions that works as the interface.
DNN	Deep Neural Network
OpenCV	Open Source Computer Vision Framework
Raspberry Pi	Small single-board computer. (For this document it will always refer to the model 3)
VizWiz	App for iPhone to assist blind people
AWS Lambda	Serverless computing platform trigger by events
AWS Rekognition	Cloud deep learning service that support object recognition and other features

Sight Hepler Overview

The overview of Sight Helper is the use of computer vision to provide guidance of the surrounding by an audio feedback. Sight Helper provide two solutions the self-contain that is limited on the computer vision model but is not internet dependent. The mobile version implements the AWS Recognition service that provide a powerful computer vision.

System Requirements for Sight Helper

The two versions of Sight Helper are similar in the requirements. However, there are some minor specification that only apply for specific version. In the Following points will present the requirements of version:

Functional Requirements for Sight Helper Self-contain Version

- Objects shall be detected even with a quarter of obstructed vision.
- Objects shall be detected with minimum 40 Lux of illumination.
- Speaker shall minimum reach 40 decibels.

Non-Functional Requirements for Sight Helper Self-contain Version

- The response time for object evaluation shall be lower than 10 seconds.
- Battery shall minimum last 4 hours.

Functional Requirements for Sight Helper Mobile Version

- Evaluation of elements in picture shall be detected with a 25% or lower of obstruction.
- Evaluation of elements in picture shall be detected with a illumination as lower of 40 Lux.
- Application shall have access to camera.
- Application shall label in comprehensive speed (125 to 150 words per minute).
- Application shall detect orientation of device.

Non-Functional Requirements for Sight Helper Mobile Version

- Feedback with labels from evaluation shall be under the 10 seconds.
- Volume shall be of 40 decibels or higher.
- Application shall be capable to handle 100 users without affecting performance.
- Application shall not keep record of taken pictures.

Performance Requirements for Sight Helper

The functioning of the Sight Helper is measure founded on the efficacy of the model of the DNN. Additional performance requirements are:

- Initial boot shall start in less than 15 seconds.
- Time of respond from the system after the initial trigger shall be equal or under 10 seconds.

Hardware Specification for Sight Helper

Self-contain Version

- Raspberry Pi B 3 as Figure 1.
    - Self-contain board that hold:
        - 1.4GHz CPU.
        - 1GB RAM.
        - WIFI, LAN and Bluetooth connections.
        - 40-pin.
        - 4 USB ports.
        - HDMI port.
        - 5V power input.
        - Audio port.

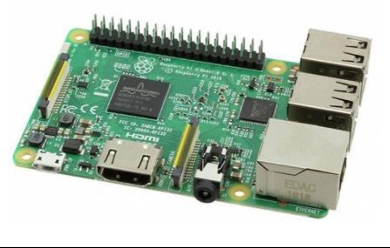


Figure 1. Raspberry Pi B 3

- Raspberry Pi Camera Board v2.1 as Figure 2.
    - 8 megapixel
    - 3280 x 2464 pixel for max image.
    - 1080p30 max resolution video.

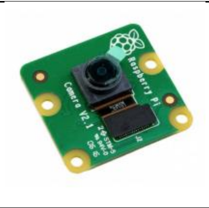


Figure 2. Raspberry Pi Camera Board v2.1

- Google Voice Kit V1 as Figure 3.
    1. HAT board
    2. Microphone board
    3. Standoffs
    4. Speaker

5. Led push button
6. 4-wire cable
7. 5-wire cable
8. Cardboard box
9. Cardboard frame

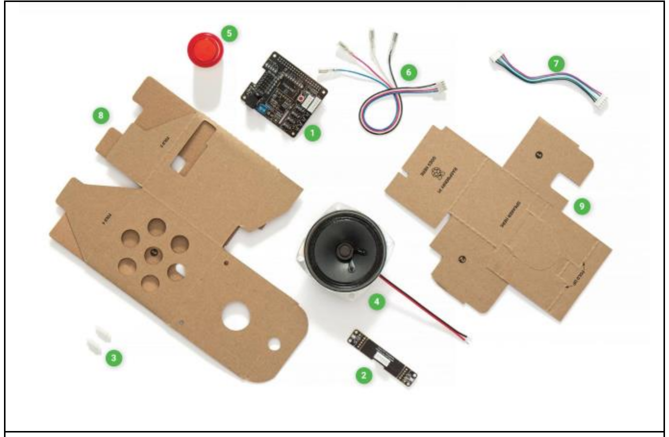


Figure 3. Google Voice Kit V1

- Micro SD Card as Figure 4.
    - The system requires a minimum 8GB. However, 16GB is recommended.

Figure 4. Micro SD Card

- USB Battery Pack as Figure 5.
    - Minimum a 5V output.
    - Minimum a 3000mAh.

Figure 5. USB Battery Pack

Mobile Version

- For IOS devices:
  - iPhone 6s or higher versions.
- For Android devices:
  - Support Android 9 or higher versions.
  - Back Camera with 6 megapixel or higher.
  - Speaker with 40 decibels or higher.

Software Specification for Sight Helper

Self-contain Version

- Raspbian
  - Linux Operating System based on Debian dedicated for the Raspberry Pi Board.
- Python 3.6
  - High level programming language.
- OpenCV for Python
  - Library open source with functions dedicated for computer vision.
- NumPy 1.16.5 or later
  - Library open source with functions dedicated for scientific and mathematic process.
- Imutils 0.5.3 or later
  - Library open source with functions dedicated for image processing.
- AIY (Drivers for Audio HAT)
  - Python drivers dedicated to handle the hardware provided in the Google AIY voice kit version 1.
- Picamera
  - Library open source with functions dedicated to handle the Pi Camera.

Mobile Version

- Internet Connection.
- Operating System:
    - IOS 13 or higher.
    - Android 9 or higher.

System Design for Sight Helper

Sight Helper Design is different in each version, the self-contain covers the hardware and software architecture. However, in the case of the mobile the system design only covers the software architecture. Both version system design is described below:

Hardware Architecture for Sight Helper Self-contain Version

Hardware Architecture is divided into three groups:

The Peripherical: These are elements that interact directly with the user. In this are the Camera, the speaker, a USB power bank, and the LED button. The Camera and the power bank are connected directly to the mainboard. However, the speaker and the LED button connected to the AIY Google Voice HAT.

The Main Board: The Main Board is the Raspberry Pi model 3. As an extension to the GPIO, the AIY Google Voice HAT which is connected directly to the GPIO of the Raspberry Pi model 3. The mainboard and the voice HAT control the peripherical, except for the power bank that works as the power source of the entire system.

The Operating System: The Operating System encapsulated in the Micro SD card. It holds the Raspbian operating system, the Object Detection code, and the drivers for the IAY peripherical.

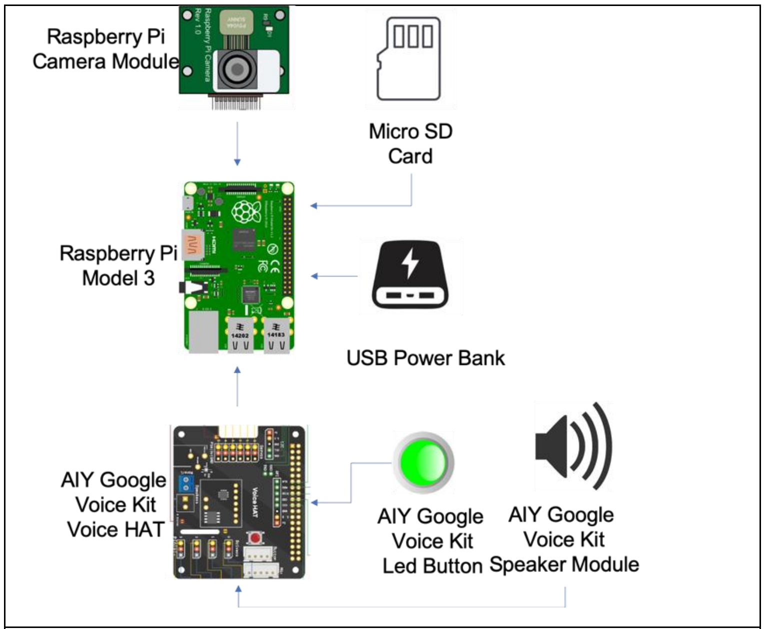

Raspberry Pi Camera Module
Micro SD Card
Raspberry Pi Model 3
USB Power Bank
AIY Google Voice Kit Voice HAT
AIY Google Voice Kit Led Button
AIY Google Voice Kit Speaker Module

Figure 6. Hardware Architecture Self-contain Version

Software Architecture for Sight Helper Self-contain Version

Software Architecture is divided into three groups:

Drivers: The Drivers are base code to control the AIY peripherical and the drivers of OpenCV that control the Computer Vision processing. The AIY drivers and the OpenCV drivers are open source.

Model: The model integrated the configuration structure and the compile DNN. This example is using the pre-train CAFEE model.

Object Recognition Code: The Object Recognition code is coded in Python 3.6 and initialize all the peripherical and the DNN model. After the initialization of the drivers. The system waits for the peripherical trigger to process and generates a loud voice the results from the DNN.

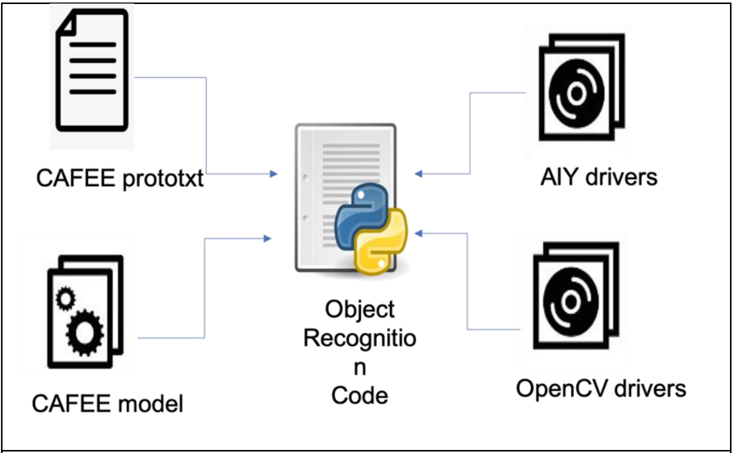


Figure 7. Software Architecture Self-contain Version

Software Architecture for Sight Helper Mobile Version

The Software Architecture of Sight Helper is divided into three services and a 3D engine:

AWS Lambda: A serverless computing platform that allows the execution of code as the response of events, it can be trigger through HTTP requests. In general AWS Lambda is a computing service that in response to events triggers the execution of code where the resources for the execution of the code are automatically managed by the service.

AWS Rekognition: A computer vision cloud-based service from Amazon. Allow the analysis of videos and images with features of detecting and comparing objects, activities, texts in the videos or images.

Google Translator: A cloud base translator for over 100 languages. It provides an instant translation of words, phrases and web sites. It also provides audio clips of the pronunciation of the phases in the corresponding languages.

Unity 3D Engine: This is considered as one of the most used game engines, top-rated game titles like Cuphead, Pokemon Go and others have been developed in Unity. It uses C# as the programming language and provides cross-platform support allowing fast development to multiple platforms with one base code.

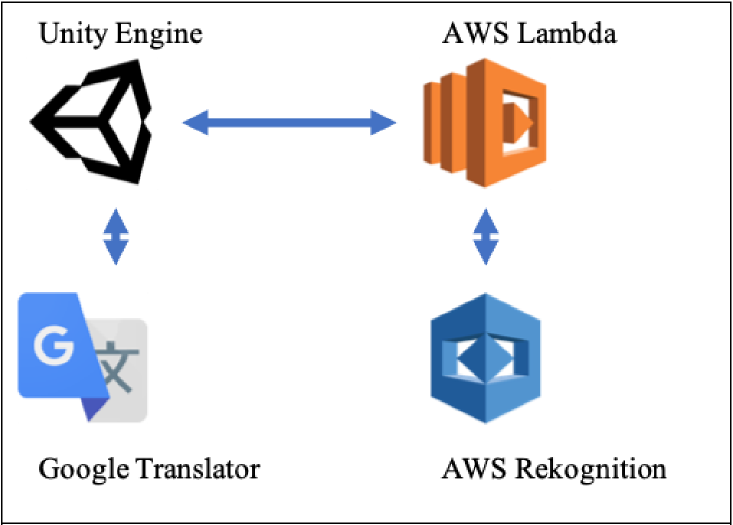
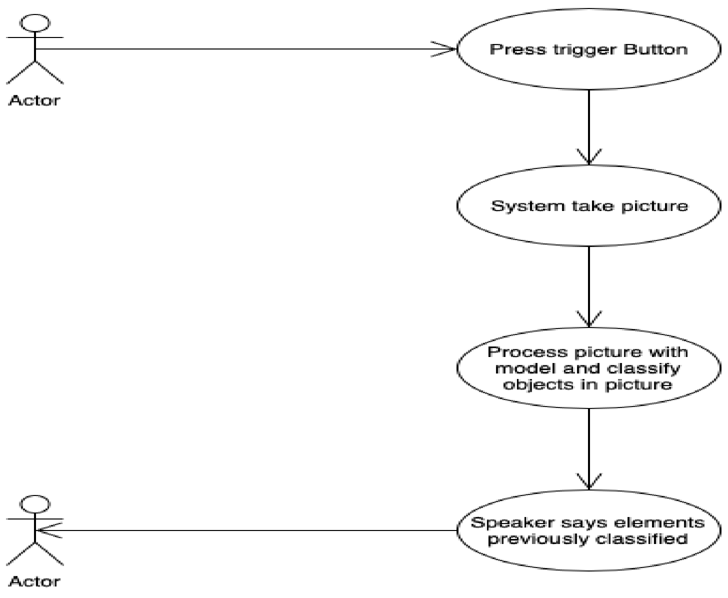


Figure 7. Software Architecture Mobile Version

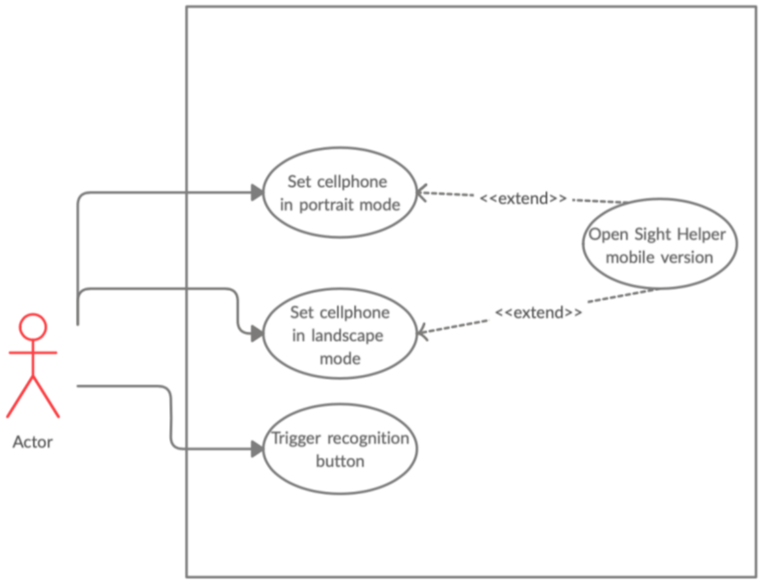
Sight Helper User Use Case

The user use case varies depending in the two versions of Sight Helper. Both versions use machine learning in the process of objects recognitions. However, each version goes through two different set of actions. In the fallowing graphs will be represented the user use case workflow for each version.

Sight Helper User Use Case Self-contain Version


Actor
Press trigger Button
System take picture
Process picture with model and classify objects in picture
Speaker says elements previously classified
Actor

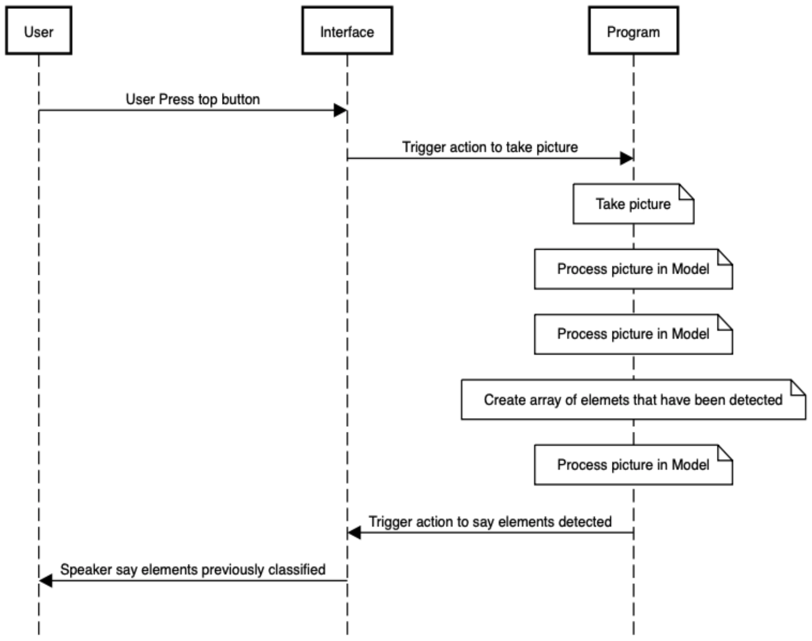
Sight Helper User Use Mobile version


Set cellphone in portrait mode
<<extend>>
Open Sight Helper mobile version
Set cellphone in landscape mode
<<extend>>
Trigger recognition button
Actor

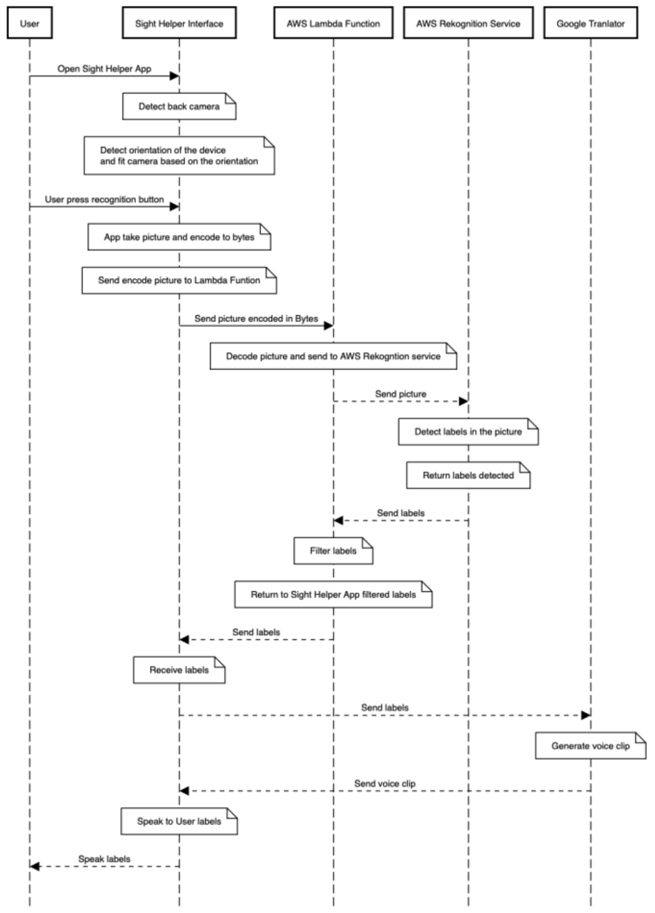
Sight Helper Sequence Diagram

The sequence diagram for Sight Helper can be divided based on the version. For the self-contain version most of all the process happen in the program flow. However, in the mobile version the process is more dived due to the multiple actors as representations of the multiple services that are been use.

Sight Helper Sequence Diagram for Self-contain Version

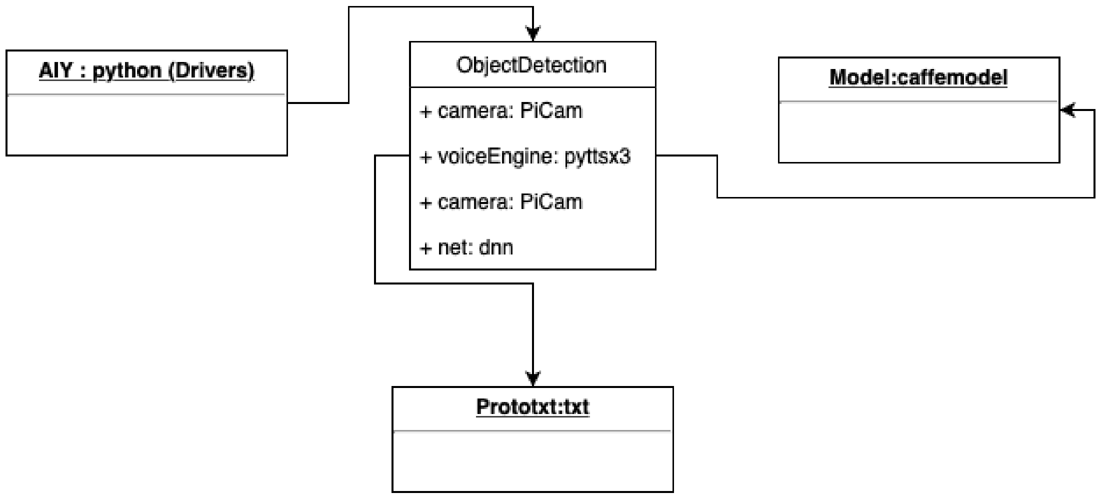

User
Interface
Program
User Press top button
Trigger action to take picture
Take picture
Process picture in Model
Process picture in Model
Create array of elemets that have been detected
Process picture in Model
Trigger action to say elements detected
Speaker say elements previously classified

Sight Helper Sequence Diagram for Mobile Version

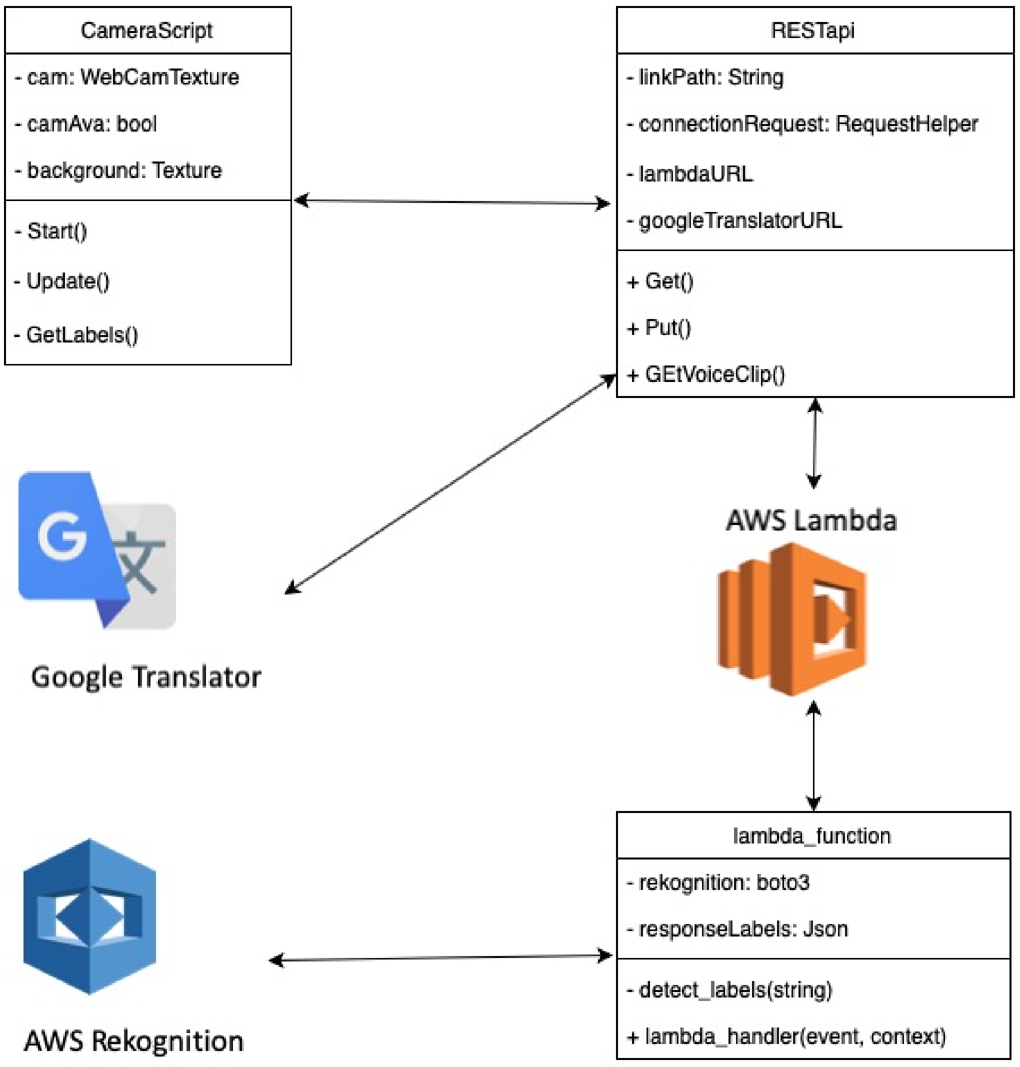

User
Sight Helper Interface
AWS Lambda Function
AWS Rekognition Service
Google Tranlator
Open Sight Helper App
Detect back camera
Detect orientation of the device and fit camera based on the orientation
User press recognition button
App take picture and encode to bytes
Send encode picture to Lambda Funtion
Send picture encoded in Bytes
Decode picture and send to AWS Rekogntion service
Send picture
Detect labels in the picture
Return labels detected
Send labels
Filter labels
Return to Sight Helper App filtered labels
Send labels
Receive labels
Send labels
Generate voice clip
Send voice clip
Speak to User labels
Speak labels

Sight Helper Class Diagram

Sight Helper Class Diagram for Self-contain Version



Sight Helper Class Diagram for Self-contain Version



Sight Helper Project Structure and Code

Sight Helper Project Structure and Code for Self-contain Version

ObjectDetection.py

```
# Instructions to use
# python3 (Object Detection Class) --protoFile (path for the
prototxt) --modelFile (path for the model)
```

```
# Example
# python3 real_time_object_detection.py --protoFile
ProtoTextConf.txt --modelFile TrainModel.caffemodel

########## Importing the necessary packages ##########

#Library with functions for image handling
import imutils

#Computer Vision Library
import cv2

#Speech Library
import pyttsx3 as se

#Library with scientific functions
import numpy as np

#Library with  command-line interfaces functions
import argparse as argp

#Library for image handling
from PIL import Image

#Google AIY drivers
from aiy.board import Board

#Library for handling PiCamera
from picamera import PiCamera
```

```
#Initialize Camera and set the resolution
camera = PiCamera(resolution = (320,240))

#Initialization of the voice engine
speakMotor = se.init()

########## Set properties of the voice engine ##########

# Speed percent (can go over 100)
speakMotor.setProperty('rate', 150)

# Volume 0-1
speakMotor.setProperty('volume', 0.9)

#Setting language of the voice (language) engine to English
languages = speakMotor.getProperty('voices')

#Looping through the languages and selecting English
for language in languages:

    print ("[INFO] "+str(language)+"\n")

    if language.languages[0] == u'en_US':
        speakMotor.setProperty('voice', language.id)
        break

########## Parsing and Creation of initial arguments ##########

argumentsParsed = argp.ArgumentParser()

#Parsing argument for the prototxt file
```

```
argumentsParsed.add_argument("-p", "--protoFile",
required=True,
    help="path for the prototxt file that is going to be
deploy")

#Parsing argument for the model file
argumentsParsed.add_argument("-m", "--modelFile",
required=True,
    help="path for the file the contain the train model")

#Parsing argument for setting the confidence level
argumentsParsed.add_argument("-c", "--confidenceLevel",
type=float, default=0.4,
    help="level of probability for the minimum detections")

#Setting dictionary of arguments for application
arguments = vars(argumentsParsed.parse_args())

########## Initializing the list of labels define in the model
##########

TRAIN_LABELS = ["bus", "car", "background",
                "aeroplane", "bicycle", "bottle",
                "cat", "chair", "dog", "horse",
                "motorbike","cow", "diningtable",
                "person", "pottedplant", "sheep",
                "sofa", "bird", "boat", "train", "tvmonitor"]

########## Initializing train model ##########

print("[INFO] Initializing train model...\n")
```

```
trainNetwork = cv2.dnn.readNetFromCaffe(arguments["protoFile"],
arguments["modelFile"])

print("[INFO] Model and libraries loaded...\n")



# loop over to process image
while True:
    #Wait for the button to trigger the camera and process the
image
    with Board() as voiceBoard:

        print('[INFO] Press button to start recording.\n')
        voiceBoard.button.wait_for_press()
        listObjects = []

        #taking photo
        camera.capture('picture.jpg')

        print("[INFO] Loading image.\n")

        # Loading image
        img = cv2.imread("picture.jpg")

        image = cv2.imread("picture.jpg")

        image = imutils.resize(image, width=400)

        # Transforming the image to blob that the Deep Neural
Network could read
```

```
        ImageBlob = cv2.dnn.blobFromImage(cv2.resize(image,
(300, 300)),
            0.007843, (300, 300), 127.5)

        # Passing the blob through the Deep Neural Network
        trainNetwork.setInput(ImageBlob)

        # Getting the predictions from the Deep Neural Network
based on the blob
        results = trainNetwork.forward()

        # Evaluating the results from the Deep Neural Network
        for i in np.arange(0, results.shape[2]):
            # Collecting the confidence level of the elements
detected by Deep Neural Network
            confidenceLevel = results[0, 0, i, 2]

            # Selecting the results that meet the confidence
level
            if confidenceLevel > arguments["confidenceLevel"]:
                # Selecting the index of the labels from the
predictions
                idx = int(results[0, 0, i, 1])

                # Appending labels of detected elements
                listObjects.append(TRAIN_LABELS[idx])

        #Say object detected
        for element in listObjects:

            print ("[INFO] "+element+"\n")
            speakMotor.say(element)
            speakMotor.runAndWait()

        if len(listObjects) == 0:
            print ("[INFO] No objects Detected.\n")
            speakMotor.say("No objects Detected")
            speakMotor.runAndWait()

#Note: Code Reference from Rosebrock 2018 [8]
```

Sight Helper Project Structure and Code for Self-contain Version

```
Lambda_function.py
# Library boto3 that enable the connection with the AWS
Rekognition Service
import boto3

# Library base64 that enable the process of the image to bytes
import base64

#Initializing the AWS Rekognition service
rekognitionService = boto3.client('rekognition')

# --------------- Main handler ------------------
def lambda_handler(event, context):

    # Initializing labels variable
    labels = ''
    # Block try that check for possible erros in the recogniton
process
    try:

        # Retrive the image from the trigger event
        imageToRecognize = event['body']

        # Decoding image from the message wrapper
        imageToRecognize = base64.b64decode(imageToRecognize)

        # Removing extra data
        imageToRecognize =
str(base64.b64decode(event['body']))[7:len(str(base64.b64decode
(event['body'])))-1]

        # Replacing codes from utf-8
        imageToRecognize = imageToRecognize.replace('%2f', '/')
        imageToRecognize = imageToRecognize.replace('%2b', '+')
        imageToRecognize = imageToRecognize.replace('%3d', '=')

        # Encoding image to bytes
        imageToRecognize = base64.b64decode(imageToRecognize)

        # Sending image for objects recognition to AWS
Rekognition Service
        recognizedObjects =
rekognitionService.detect_labels(Image =
{'Bytes':imageToRecognize},MaxLabels=10)

        # Merging all the labels in one string
        for label in recognizedObjects['Labels']:
            labels = labels +' '+label['Name']

    # Exception handler in case of error in the recognition
process
    except:
        response = None
        labels = "Error on recognition"

    # Returning json object with the success code and the
labels of the objects recognized
    return {
        'statusCode': 200,
        'body': labels
    }
```

CameraScript.cs

```
// System pakages
using UnityEngine;
using UnityEngine.UI;

// This Class handle the orientation of the camera based on the
orientation of phone
public class CameraScript : MonoBehaviour
{
    // Initial variables
    private WebCamTexture backCamera;
    private bool cameraAvailable;
    private RawImage imageBackground;
    private AspectRatioFitter ratioFit;
    // Function the automatically is executed and define the
back camera
    private void Start()
    {
        // Looking for possible cameras on the system
        WebCamDevice[] cameraList = WebCamTexture.devices;

        // Check if there are camera in the system
        if (cameraList.Length == 0)
        {
            Debug.Log("No camera detected");
            return;
        }

        // Loop through the possible cameras to set the back
camera
        for (int i =0; i< cameraList.Length;i++)
        {
            if(!cameraList[i].isFrontFacing)
            {
                backCamera = new
WebCamTexture(cameraList[i].name, Screen.width, Screen.height);
            }
        }

        // Check if a camera was select as the back camera
        if(backCamera == null)
        {
            Debug.Log("Unable to find back camera");
            return;
        }
        // Activate the back camera
        backCamera.Play();

        // Set the texture of the background as the back camera
view
        imageBackground.texture = backCamera;

        // Activate variable that check if a camera is set up
        cameraAvailable = true;
    }

    //  Function that is called once per frame and fix the
camera based on the orientation of the phone
    private void Update()
    {
        // Check if a camera is set up
        if (!cameraAvailable)
            return;

        // Update the ratio based on the orientation of the
phone
        float ratio = (float)backCamera.width /
(float)backCamera.height;
        ratioFit.aspectRatio = ratio;

        // Update the Y scale based on the orientation of the
phone
        float scaleY = backCamera.videoVerticallyMirrored ? -1f
: 1f;
```

```
        imageBackground.rectTransform.localScale = new
Vector3(1f, scaleY, 1f);

        // Update the orientation based on the orientation of
the phone
        int orientation = -backCamera.videoRotationAngle;
        imageBackground.rectTransform.localEulerAngles = new
Vector3(0, 0, orientation);
    }
}
```

```
RESTapi.cs
```

```
// System pakages
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Text;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;
// This Class handle the process of taking the photo and
interact with the recognition service and voice generator
public class RESTapi : MonoBehaviour
{
    // Initial variables
    private AudioSource audioSource;
    private RawImage background;
    private String labels;
    // Function the automatically is executed and define the
audio source
```

```
    void Start()
    {
        audioSource = gameObject.GetComponent<AudioSource>();
    }
    // Function that is trigger by the front button on the app
    public void Recognition()
    {
        // It execute the request to the AWS Lambda function
        StartCoroutine(GetRequest("https://7fyk5orvj6.execute-
api.us-east-2.amazonaws.com/default/recogntion"));
    }
    // Function the request the audio clip with the labels of
the objects recognized on the picture
    IEnumerator GetAudioClip()
    {
        // Execute a web request with the labels of the objects
recognized on the picture
        using (UnityWebRequest webRequest =
UnityWebRequestMultimedia.GetAudioClip("https://translate.googl
e.com/" +
            "translate_tts?ie=UTF-
8&total=1&idx=0&textlen=32&client=tw-ob&q="+labels+"&tl=En-gb",
AudioType.MPEG))
        {
            yield return webRequest.SendWebRequest();
            if (webRequest.isNetworkError)
            {
                Debug.Log(webRequest.error);
            }
            else
            {
                // Download the audio clip of the labels
                AudioClip audioClip =
DownloadHandlerAudioClip.GetContent(webRequest);
                // Setting the clip in the audio source
                audioSource.clip = audioClip;
                // Playing audio clip
                audioSource.Play();
            }
        }
    }
    // Function the request the labels of the objects
recognized on the picture
    IEnumerator GetRequest(string uri)
    {
        // Initial web form
        WWWForm webForm = new WWWForm();
        // Path to the picture
        string filename = "picture.png";
        string pictureDirectory =
Path.Combine(Application.persistentDataPath, filename);
        // Creating screenshot
        ScreenCapture.CaptureScreenshot("picture.png");
        // Reading picture from file
        byte[] file = File.ReadAllBytes(pictureDirectory);
        // Convert picture to string
        string base64String = Convert.ToBase64String(file);
        // Setting the form with the picture
        webForm.AddField("body", base64String);
        // Execute a web request with the form
        using (UnityWebRequest webRequest =
UnityWebRequest.Post(uri, webForm))
```

```
        {
            // Wait for result of the object recognition
service.
            yield return webRequest.SendWebRequest();
            string[] pages = uri.Split('/');
            int page = pages.Length - 1;
            if (webRequest.isNetworkError)
            {
                Debug.Log(pages[page] + ": Error: " +
webRequest.error);
            }
            else
            {
                StringBuilder sb = new StringBuilder();
                foreach
(System.Collections.Generic.KeyValuePair<string, string> dict
in webRequest.GetResponseHeaders())
                {
                    sb.Append(dict.Key).Append(":
\t[").Append(dict.Value).Append("]\n");
                }

                Dictionary<string, string> responseHeaders =
webRequest.GetResponseHeaders();
                labels = webRequest.downloadHandler.text;
                StartCoroutine(GetAudioClip());
            }
        }
    }
}
```

Sight Helper Test Cases

Sight Helper Test Cases for Self-contain Version

Object Recognition Feature Success

Test Case Number:	TC1
Revision:	Rev. 1
Author:	Misael Uzcategui
Date Conducted:	Oct 18, 2019
Test Conductor:	Misael Uzcategui
Customer Representative:	Dr. Abrar Qureshi
Description:	The present Test Case will be testing the feature of object recognition for Sight Helper. UC1 is the use case of reference.
Pre-Test Setup:	1. Ensure the Battery pack is charge. 2. System is fully installed in the Raspberry Pi. 3. At least one of the elements facing the camera should be part of the pre-train set.

Use Case: 01 Flow: Main Flow

User Action	Expected Results	Pass/Fail	Comment
1. A user should plug the USB from the battery pack to power connector in the raspberry Pi.	Red Light should light up.	Pass	
2. User should wait 10 seconds to load system.		Pass	
3. User should face the camera to the area where are the objects that are going to be recognized.		Pass	
4. User press button on the top of the Sight Helper.	After 8 -10 seconds Speaker should list the elements that where recognized.	Pass	
Post-conditions: 1. User should unplug the USB cable from the Raspberry Pi.			

Object Recognition Feature Fail

Test Case Number:	TC2
Revision:	Rev. 1
Author:	Misael Uzcategui
Date Conducted:	Oct 18, 2019
Test Conductor:	Misael Uzcategui
Customer Representative:	Dr. Abrar Qureshi
Description:	The present Test Case will be testing the feature of object recognition from Sight Helper. UC1 is the use case of reference.
Pre-Test Setup:	1. Ensure the Battery pack is charge. 2. System is fully installed in the Raspberry Pi. 3. No elements of the pre-train set should be facing the camera.

Use Case: 01 Flow: Alternative

User Action	Expected Results	Pass/Fail	Comment
1. A user should plug the USB from the battery pack to power connector in the raspberry Pi.	Red Light should light up.	Pass	
2. User should wait 10 seconds to load system.		Pass	
3. User should face the camera to the area where are the objects that are going to be recognized.		Pass	
4. User press button on the top of the Sight Helper.	After 8 -10 seconds Speaker should say “No object detected”.	Pass	
Post-conditions: 1. User should unplug the USB cable from the Raspberry Pi.			

Sight Helper Test Cases for Mobile Version

Camera Orientation

Test Case Number:	Test Case 1
Revision:	Rev. 1
Author:	Misael Uzcategui
Date Conducted:	Feb 20, 2020
Test Conductor:	Misael Uzcategui
Customer Representative:	Dr. Abrar Qureshi
Description:	The Camera Orientation Test Case test the functionality of reset the orientation of the camera-based cellphone orientation while using Sight Helper mobile version. User Case 1 (UC1) is the use case related for this specific test case.
Pre-Test Setup:	1. Guarantee the cellphone is charge. 2. Ensure there is no obstruction in front of the camera. 3. Camera should be facing recognizable elements.

Use Case: 01 Flow: Main Flow

User Action	Expected Results	Pass/Fail	Comment
5. User should already have install the Sight Helper mobile version App.	Sight Helper mobile version icon in the list of apps.	Pass	
6. User should open the Sight Helper mobile version App.	Display Helper mobile version App graphic interface.	Pass	
7. User should set the cellphone in portrait position.	Display should adjust to portrait mode.	Pass	
8. User should set the cellphone in landscape mode.	Display should adjust to landscape mode.	Pass	
Post-conditions: 1. No applicable.			

Trigger Recognition with Internet

Test Case Number:	Test Case 2
Revision:	Rev. 1
Author:	Misael Uzcategui
Date Conducted:	Feb 20, 2020
Test Conductor:	Misael Uzcategui
Customer Representative:	Dr. Abrar Qureshi
Description:	The Trigger Recognition Test Case test the functionality of recognition of elements using the Sight Helper mobile version. User Case 1 (UC1) is the use case related for this specific test case.
Pre-Test Setup:	1. Guarantee the cellphone is charge. 2. Ensure there is no obstruction in front of the camera. 3. Camera should be facing recognizable elements. 4. Guarantee internet access.

Use Case: 01 Flow: Main Flow

User Action	Expected Results	Pass/Fail	Comment
1. User should already have installed the Sight Helper mobile version App.	Sight Helper mobile version icon in the list of apps.	Pass	
2. User should open the Sight Helper mobile version App.	Display Helper mobile version App graphic interface.	Pass	
3. User should press the recognition button with the cellphone camera facing the objects that should be recognized.	Sight Helper mobile version should play a clip with the labels of the objects previously recognized.	Pass	
Post-conditions: 1. No applicable.			

Trigger Recognition without Internet

Test Case Number:	Test Case 3
Revision:	Rev. 1
Author:	Misael Uzcategui
Date Conducted:	Feb 20, 2020
Test Conductor:	Misael Uzcategui
Customer Representative:	Dr. Abrar Qureshi
Description:	The Trigger Recognition Test Case test the functionality of recognition of elements using the Sight Helper mobile version without internet access. User Case 1 (UC1) is the use case related for this specific test case.
Pre-Test Setup:	1. Guarantee the cellphone is charge. 2. Ensure there is no obstruction in front of the camera. 3. Camera should be facing recognizable elements. 4. Guarantee no internet access.

Use Case: 01 Flow: Main Flow

User Action	Expected Results	Pass/Fail	Comment
1. User should already have installed the Sight Helper mobile version App.	Sight Helper mobile version icon in the list of apps.	Pass	
2. User should open the Sight Helper mobile version App.	Display Helper mobile version App graphic interface.	Pass	
3. User should press the recognition button with the cellphone camera facing the objects that should be recognized.	Sight Helper mobile version should play a clip that say “Unable to do recognition test internet access”.	Pass	
Post-conditions: 1. No applicable.			

Ethical and Societal Effect of Sight Helper

Any software and hardware are responsible for certain ethical and social effect, and by kipping this idea in mind every development should be aware and contemplate the haw and what type of effect will create the use and implementation it. In the case of Sight Helper are being considered the possible effects as the responsibilities that have accept. However, is important to consider some exception like in the case of the mobile version since certain elements are out of control, such as the device in which it is used.

Environmental Responsibility:

The environmental responsibility is only consider to the Sight Helper Self-contain Version given that the mobile version depend on the device where is install.

- The components related to the elaboration of Sight Helper Self-contain Version shall be recyclable.
- As part of the compromise to reduce carbon print Sight Helper Self-contain shall work with low voltage to engage clean energy as power source.

Technical Responsibility:

- Self-contain and mobile version of Sight Helper shall give credit and respect the copyright of the patents and algorithms used.
- The testing process for Sight Helper shall be done without bias and with transparency.
- The hardware and software of Sight Helper shall be clear on any malware or any inside code that affect the performance.
- The hardware and software of Sight Helper shall fallow and satisfy the protocols and standards of the industry.

Customer Responsibility:

- Sight Helper shall treat all data related to the user carefully and with responsibility.
- Any outside contribution or citation in any of the elements of Sight Helper shall be documented and fallow the licenses and property rights associated to each element.
- Any reproduction or modification on the code or design of Sight Helper in any of their versions shall be subject to prior approval.

- All documentation and code of Sight Helper in any of their versions shall be according to the industry standards.

Sight Helper Limitations

Limitations in Sight Helper Self-contain Version

1. Limited amount of object recognition.
2. Possible overfitting of the model.
3. No user-friendly option to update the model.
4. No depth supports.

Limitations in Sight Helper Self-contain Version

1. Limited by internet connection.
2. Limited by the AWS Rekognition model.
3. No depth supports.

Sight Helper Future Work

There a significant number of features that can be add to the Sight Helper app in order to provide more complete guidance to user. The following list is features that could be cover in future versions:

- Use of GPS to provide information of the geolocation and reference of the surrounding.
- Use of voice command to provide weather information.
- Use of voice command to provide estimates of time and path reference between two landmarks.
- Offline backup recognition model as a contingency plan in case internet access is lost.

Bibliography

1. Abaya, W. F., Basa, J., Sy, M., Abad, A. C., & Dadios, E. P. (2014). Low cost smart security camera with night vision capability using Raspberry Pi and OpenCV. *2014 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*. doi: 10.1109/hnicem.2014.7016253

2. Bhanse, V. K., & Jaybhaye, M. (2018). Face Detection and Tracking Using Image Processing on Raspberry Pi. *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*. doi: 10.1109/icirca.2018.8597246

3. Cameron, K., & Islam, M. S. (2018). Multiple Objects Detection using HSV. *NAECON 2018 - IEEE National Aerospace and Electronics Conference*. doi: 10.1109/naecon.2018.8556711

4. Chang, O. (2014). Reliable object recognition by using cooperative neural agents. *2014 International Joint Conference on Neural Networks (IJCNN)*. doi: 10.1109/ijcnn.2014.6889412

5. Guennouni, S., Ahaitouf, A., & Mansouri, A. (2014). Multiple object detection using OpenCV on an embedded platform. *2014 Third IEEE International Colloquium in Information Science and Technology (CIST)*. doi: 10.1109/cist.2014.7016649

6. Kaymak, C., & Ucar, A. (2018). Implementation of Object Detection and Recognition Algorithms on a Robotic Arm Platform Using Raspberry Pi. *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*. doi: 10.1109/idap.2018.8620916

7. Mazumdar, M., Sarasvathi, V., & Kumar, A. (2017). Object recognition in videos by sequential frame extraction using convolutional neural networks and fully connected neural

networks. *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*. doi: 10.1109/icecds.2017.8389692

8. Rosebrock, A. (2018, November 25). Raspberry Pi: Deep learning object detection with OpenCV. Retrieved from https://www.pyimagesearch.com/2017/10/16/raspberry-pi-deep-learning-object-detection-with-opencv/.

9. Salih, F., & Omer, S. M. (2018). Raspberry pi as a Video Server. *2018 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*. doi: 10.1109/iccceee.2018.8515817

10. Shah, M., & Kapdi, R. (2017). Object detection using deep neural networks. *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*. doi: 10.1109/iccons.2017.8250570

11. Tepelea, L., Buciu, I., Grava, C., Gavrilut, I., & Gacsadi, A. (2019). A Vision Module for Visually Impaired People by Using Raspberry PI Platform. *2019 15th International Conference on Engineering of Modern Electric Systems (EMES)*. doi: 10.1109/emes.2019.8795205

12. Bagwan, S. M. R., & Sankpal, L. J. (2015). VisualPal: A mobile app for object recognition for the visually impaired. *2015 International Conference on Computer, Communication and Control (IC4)*. doi: 10.1109/ic4.2015.7375665

13. Brady, E., Morris, M. R., Zhong, Y., White, S., & Bigham, J. P. (2013). Visual challenges in the everyday lives of blind people. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* - *CHI 13*. doi: 10.1145/2470654.2481291

14. Chen, H.-E., Lin, Y.-Y., Chen, C.-H., & Wang, I.-F. (2015). BlindNavi. *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems* - *CHI EA 15*. doi: 10.1145/2702613.2726953

15. Ghidini, E., Almeida, W. D. L., Manssour, I. H., & Silveira, M. S. (2016). Developing Apps for Visually Impaired People: Lessons Learned from Practice. *2016 49th Hawaii International Conference on System Sciences (HICSS)*. doi: 10.1109/hicss.2016.704

16. Lin, W.-T., Krintz, C., Wolski, R., Zhang, M., Cai, X., Li, T., & Xu, W. (2018). Tracking Causal Order in AWS Lambda Applications. *2018 IEEE International Conference on Cloud Engineering (IC2E)*. doi: 10.1109/ic2e.2018.00027

17. Prudtipongpun, V., Buakeaw, W., Rattanapongsen, T., & Sivaraksa, M. (2015). Indoor Navigation System for Vision-Impaired Individual: An Application on Android Devices. *2015 11th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*. doi: 10.1109/sitis.2015.66

18. Spichkova, M., Zyl, J. V., Sachdev, S., Bhardwaj, A., & Desai, N. (2019). Easy Mobile Meter Reading for Non-smart Meters: Comparison of AWS Rekognition and Google Cloud Vision Approaches. *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering*. doi: 10.5220/0007762301790188

19. Temel, D., Lee, J., & Alregib, G. (2018). CURE-OR: Challenging Unreal and Real Environments for Object Recognition. *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. doi: 10.1109/icmla.2018.00028

20. Villamizar, M., Garces, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., … Lang, M. (2016). Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures. *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. doi: 10.1109/ccgrid.2016.37

21. Zhong, Y., Garrigues, P. J., & Bigham, J. P. (2013). Real time object scanning using a mobile phone and cloud-based visual search engine. *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility - ASSETS 13*. doi: 10.1145/2513383.2513443