DKT 221

OPERATING SYSTEM

PROJECT TITLE:

GOOGLE'S AI SMART ASSISTANT USING PI

NO	NAMA AHLI KUMPULAN	NO. MATRIK
1.	FAKHRUL MUKMIN BIN MANSOR	182020041
2.	ARVIND A/L ELLANGOPAN	182021255
3.	FAIDZRUL ADZFAR BIN FAKHRUL RADZI	182020040
4.	AKMAL HIDAYAT BIN ASRUL EFFENDI	182020022
5.	AMIRULL HAKEEM BIN MOHD NAJIB	182020023

INTRODUCTION

Artificial intelligence (AI) is the general study of making intelligent machines. Machine learning (ML), a subset of AI, focuses on the ability of machines to receive data and learn for themselves without being programmed with rules. ML differs from traditional programming by allowing you to teach your program with examples rather than a list of instructions. Instead of writing instructions, or rules, while programming, machine learning enables you to "train" an algorithm so that it can learn on its own, and then adjust and improve as it learns more about the information it is processing. In our project , we made a Google's AI Smart Assistant using Raspberry Pi as an intelligent machines to ease human daily life. This gadget will become an integral and intimate part of everyday life for millions of people in Earth. Today, developments are rapidly under way to take this phenomenon an important step further,so as a team of Unisoft Sdn Bhd we create an AI assistant to help human in their daily life. Welcome to Internet Of Things.

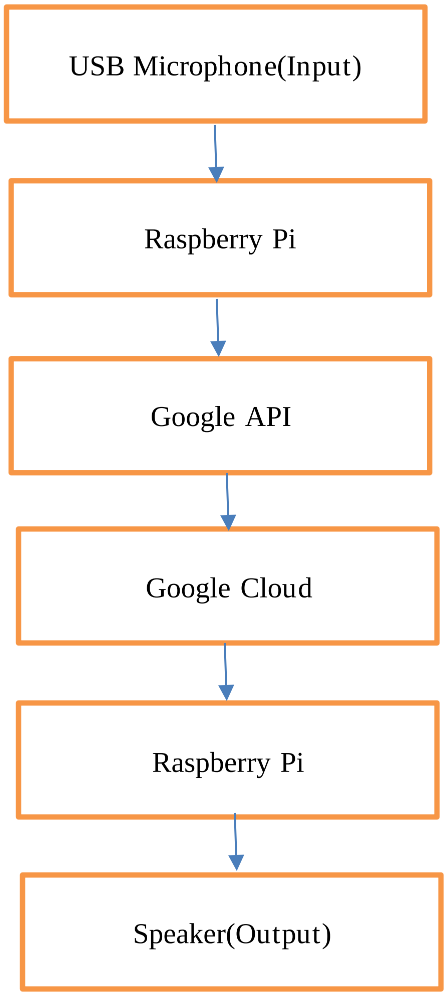
OBJECTIVES

- Learn the operation of Raspberry Pi in making a lot of technologies.
- To know the the general study of making intelligent machines
- Help human in their daily life.
- Artificial intelligence (AI) can learn a lot of things by human command and their behavior.

PROBLEM STATEMENT

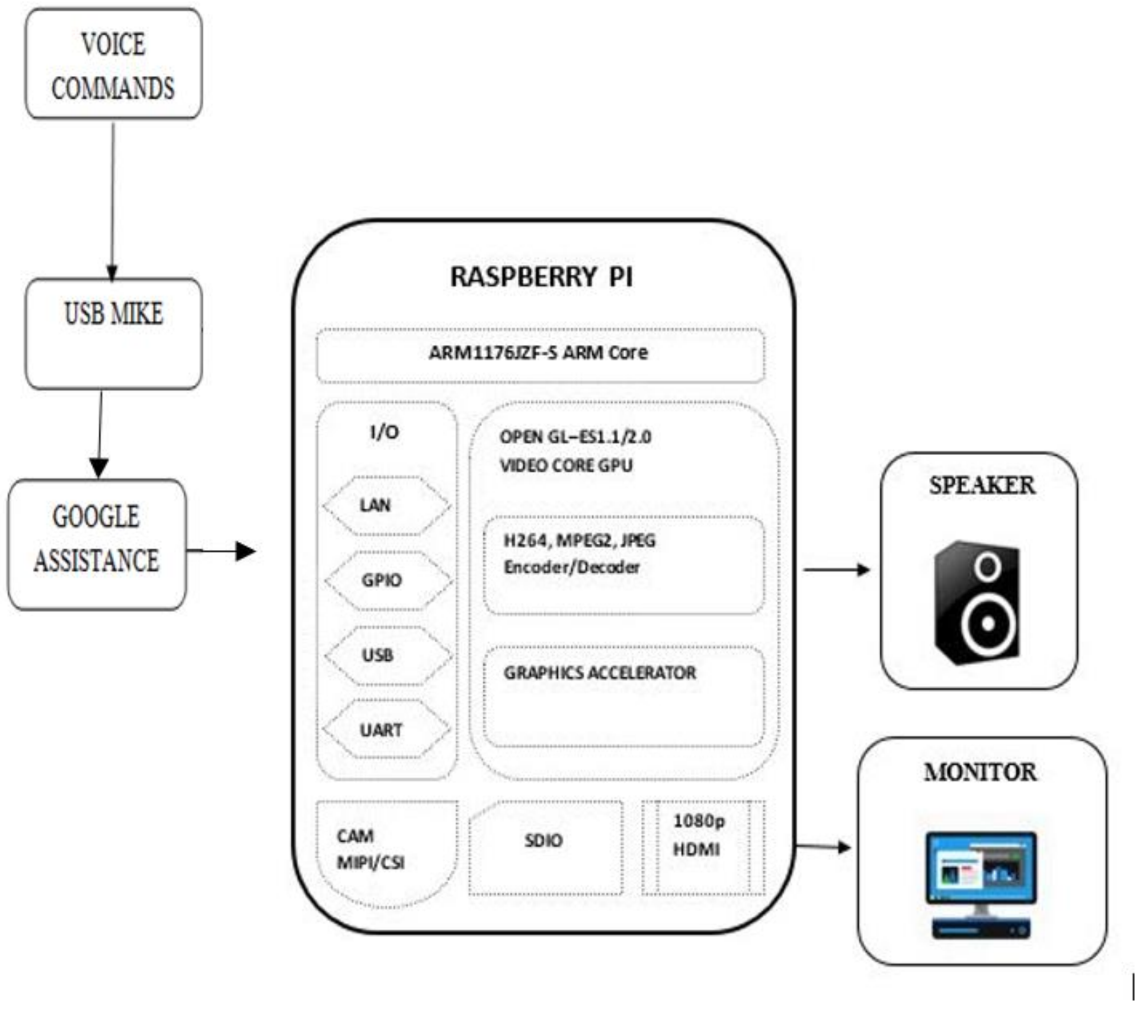
- The voice cant detect different command and some ascent, and cant detect different languages except we setting it for.
- Furthermore, we have some difficulties when our output was not in our expectations because of the internet and some bog and errors. But, we try our best to get the output with changing our coding and get a better wire and jumper .
- In future,we will add more features to our AI assistant speaker and add devices that can be controlled by the Assistant and we are looking forward and working hard to launch our upcoming project, Google Home 2.0.

METHODOLOGY



Start to talk to the Google Assistant by saying "Hey Google.". Then input from your voice going through USB Microphone and going to Raspberry Pi and Raspberry Pi send the input to the Google API to recognize your command and voices. Google API send the data to the Google cloud and cloud send the data to the Raspberry Pi to make the output in speaker.

BLOCK DIAGRAM



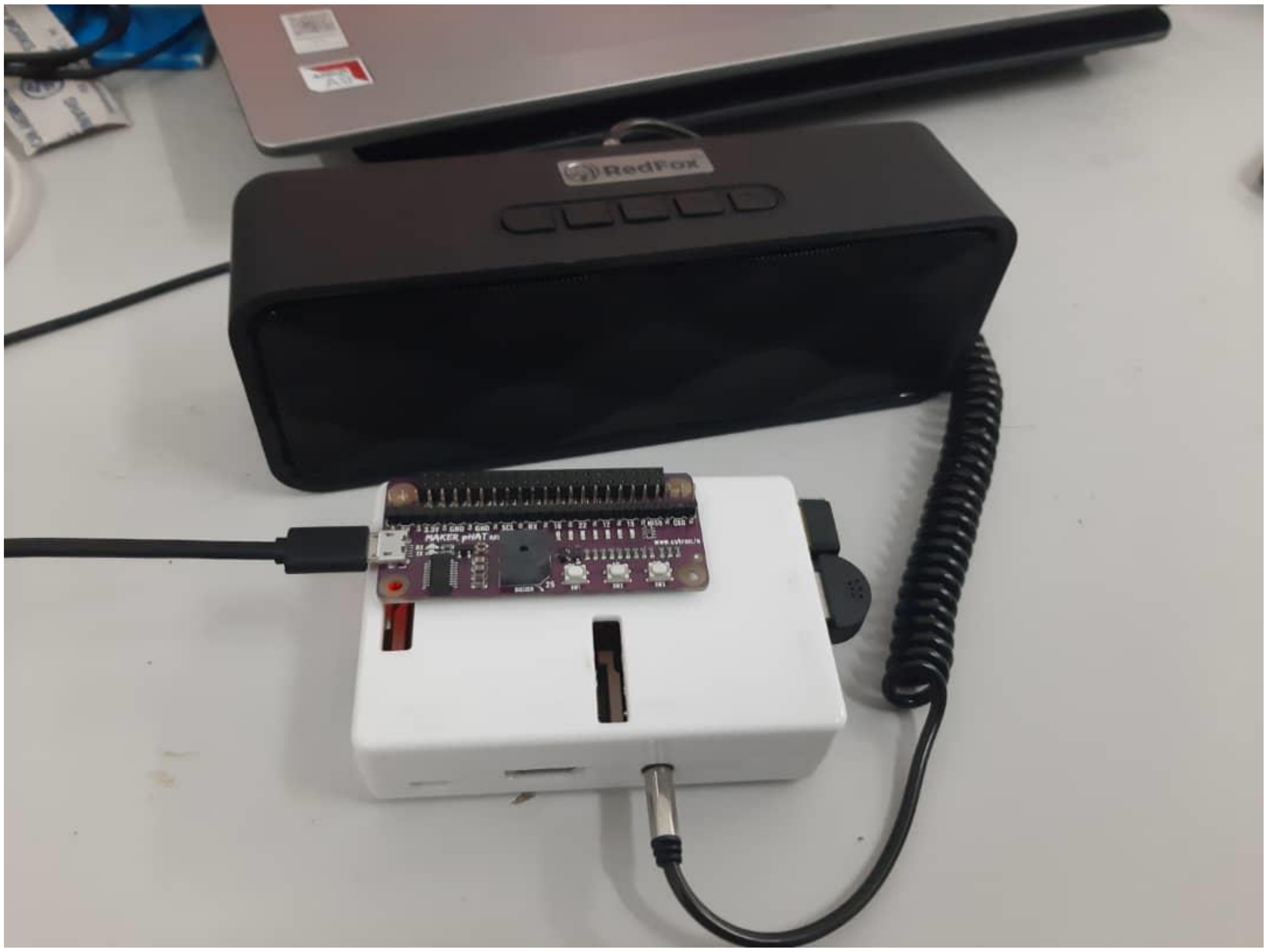
EXPLANATION FOR THE BLOCK DIAGRAM

.USB MIC- Is an easy way of making high quality recordings on your computer.USB Mics are highly portable and work on PC, Mac, iPad.Since there is only one aux input in pi,we used a usb mic

.SPEAKER -Speaker is an output hardware device that connects to a computer to generate soun d.The output is generated via speaker in audion form

.GOOGLE ASSISTANCE-Google Assistance is the AI software that is installed in this PI.The assistance operates with the the help of Google API and Google Cloud

DETAIL CIRCUIT



SOURCE CODE

1.To configure speaker and mic

```
pcm.!default {
  type asym
  capture.pcm "mic"
  playback.pcm "speaker"
}
pcm.mic {
  type plug
  slave {
    pcm "hw:1,0"
  }
}
pcm.speaker {
  type plug
  slave {
    pcm "hw:0,0"
  }
}
```

2.To install Google Assistant in Pi

```
mkdir ~/googleassistant
nano ~/googleassistant/credentials.json
```

3.To install python in Pi

```
sudo apt-get install python3-dev python3-venv libssl-dev libffi-dev libportaudio2
```

4.To install Google Assistant Library

```
python3 -m pip install --upgrade google-assistant-library
python3 -m pip install --upgrade google-assistant-sdk[samples]
```

5.To authorize Raspberry Pi for the Google Assistant

```
googlesamples-assistant-pushtotalk --project-id <projectid> --device-model-id <deviceid>
```

6.To activate python environment

```
source env/bin/activate
```

7.To use Google Assistant in Pi

```
googlesamples-assistant-pushtotalk --project-id my-dev-project --device-model-id my-model
```

8.Main coding of file Google Assistant

```
# Copyright (C) 2017 Google Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""Sample that implements a gRPC client for the Google Assistant API."""

import concurrent.futures
import json
import logging
import os
import os.path
import pathlib2 as pathlib
import sys
import time
import uuid

import click
import grpc
import google.auth.transport.grpc
import google.auth.transport.requests
import google.oauth2.credentials

from google.assistant.embedded.v1alpha2 import (
    embedded_assistant_pb2,
    embedded_assistant_pb2_grpc
)
from tenacity import retry, stop_after_attempt, retry_if_exception

try:
    from . import (
        assistant_helpers,
        audio_helpers,
        browser_helpers,
        device_helpers
    )
except (SystemError, ImportError):
    import assistant_helpers
    import audio_helpers
    import browser_helpers
    import device_helpers
```

```
ASSISTANT_API_ENDPOINT  = 'embeddedassistant.googleapis.com'
END_OF_UTTERANCE  = embedded_assistant_pb2.AssistResponse.END_OF_UTTERANCE
DIALOG_FOLLOW_ON  = embedded_assistant_pb2.DialogStateOut.DIALOG_FOLLOW_ON
CLOSE_MICROPHONE  = embedded_assistant_pb2.DialogStateOut.CLOSE_MICROPHONE
PLAYING = embedded_assistant_pb2.ScreenOutConfig.PLAYING
DEFAULT_GRPC_DEADLINE  = 60 * 3 + 5


class SampleAssistant(object):
    """Sample Assistant that supports conversations and device actions.
    Args:
      device_model_id: identifier of the device model.
      device_id: identifier of the registered device instance.
      conversation_stream(ConversationStream): audio stream
        for recording query and playing back assistant answer.
      channel: authorized gRPC channel for connection to the
        Google Assistant API.
      deadline_sec: gRPC deadline in seconds for Google Assistant API call.
      device_handler: callback for device actions.
    """

    def __init__(self, language_code, device_model_id, device_id,
                 conversation_stream, display,
                 channel, deadline_sec, device_handler):
        self.language_code = language_code
        self.device_model_id = device_model_id
        self.device_id = device_id
        self.conversation_stream = conversation_stream
        self.display = display

        # Opaque blob provided in AssistResponse that,
        # when provided in a follow-up AssistRequest,
        # gives the Assistant a context marker within the current state
        # of the multi-Assist()-RPC "conversation".
        # This value, along with MicrophoneMode, supports a more natural
        # "conversation" with the Assistant.
        self.conversation_state = None
        # Force reset of first conversation.
        self.is_new_conversation = True

        # Create Google Assistant API gRPC client.
        self.assistant = embedded_assistant_pb2_grpc.EmbeddedAssistantStub(
            channel
        )
        self.deadline = deadline_sec

        self.device_handler = device_handler

    def __enter__(self):
        return self

    def __exit__(self, etype, e, traceback):
        if e:
            return False
        self.conversation_stream.close()
```

```
def is_grpc_error_unavailable(e):
    is_grpc_error = isinstance(e, grpc.RpcError)
    if is_grpc_error and (e.code() == grpc.StatusCode.UNAVAILABLE):
        logging.error('grpc unavailable error: %s', e)
        return True
    return False

@retry(reraise=True, stop=stop_after_attempt(3),
       retry=retry_if_exception(is_grpc_error_unavailable))
def assist(self):
    """Send a voice request to the Assistant and playback the response.
    Returns: True if conversation should continue.
    """
    continue_conversation = False
    device_actions_futures = []

    self.conversation_stream.start_recording()
    logging.info('Recording audio request.')

    def iter_log_assist_requests():
        for c in self.gen_assist_requests():
            assistant_helpers.log_assist_request_without_audio(c)
            yield c
        logging.debug('Reached end of AssistRequest iteration.')

    # This generator yields AssistResponse proto messages
    # received from the gRPC Google Assistant API.
    for resp in self.assistant.Assist(iter_log_assist_requests(),
                                      self.deadline):
        assistant_helpers.log_assist_response_without_audio(resp)
        if resp.event_type == END_OF_UTTERANCE:
            logging.info('End of audio request detected.')
            logging.info('Stopping recording.')
            self.conversation_stream.stop_recording()
        if resp.speech_results:
            logging.info('Transcript of user request: "%s".',
                         ' '.join(r.transcript
                                  for r in resp.speech_results))
        if len(resp.audio_out.audio_data) > 0:
            if not self.conversation_stream.playing:
                self.conversation_stream.stop_recording()
                self.conversation_stream.start_playback()
                logging.info('Playing assistant response.')
            self.conversation_stream.write(resp.audio_out.audio_data)
        if resp.dialog_state_out.conversation_state:
            conversation_state = resp.dialog_state_out.conversation_state
            logging.debug('Updating conversation state.')
            self.conversation_state = conversation_state
        if resp.dialog_state_out.volume_percentage != 0:
            volume_percentage = resp.dialog_state_out.volume_percentage
            logging.info('Setting volume to %s%%', volume_percentage)
            self.conversation_stream.volume_percentage = volume_percentage
        if resp.dialog_state_out.microphone_mode == DIALOG_FOLLOW_ON:
            continue_conversation = True
            logging.info('Expecting follow-on query from user.')
```

```
elif resp.dialog_state_out.microphone_mode == CLOSE_MICROPHONE:
            continue_conversation = False
        if resp.device_action.device_request_json:
            device_request = json.loads(
                resp.device_action.device_request_json
            )
            fs = self.device_handler(device_request)
            if fs:
                device_actions_futures.extend(fs)
        if self.display and resp.screen_out.data:
            system_browser = browser_helpers.system_browser
            system_browser.display(resp.screen_out.data)

    if len(device_actions_futures):
        logging.info('Waiting for device executions to complete.')
        concurrent.futures.wait(device_actions_futures)

    logging.info('Finished playing assistant response.')
    self.conversation_stream.stop_playback()
    return continue_conversation

def gen_assist_requests(self):
    """Yields:  AssistRequest messages to send to the API."""

    config = embedded_assistant_pb2.AssistConfig(
        audio_in_config=embedded_assistant_pb2.AudioInConfig(
            encoding='LINEAR16',
            sample_rate_hertz=self.conversation_stream.sample_rate,
        ),
        audio_out_config=embedded_assistant_pb2.AudioOutConfig(
            encoding='LINEAR16',
            sample_rate_hertz=self.conversation_stream.sample_rate,
            volume_percentage=self.conversation_stream.volume_percentage,
        ),
        dialog_state_in=embedded_assistant_pb2.DialogStateIn(
            language_code=self.language_code,
            conversation_state=self.conversation_state,
            is_new_conversation=self.is_new_conversation,
        ),
        device_config=embedded_assistant_pb2.DeviceConfig(
            device_id=self.device_id,
            device_model_id=self.device_model_id,
        )
    )
    if self.display:
        config.screen_out_config.screen_mode = PLAYING
    # Continue current conversation with later requests.
    self.is_new_conversation = False
    # The first AssistRequest must contain the AssistConfig
    # and no audio data.
    yield embedded_assistant_pb2.AssistRequest(config=config)
    for data in self.conversation_stream:
        # Subsequent requests need audio data, but not config.
        yield embedded_assistant_pb2.AssistRequest(audio_in=data)
```

```
@click.command()
@click.option('--api-endpoint', default=ASSISTANT_API_ENDPOINT,
              metavar='<api endpoint>', show_default=True,
              help='Address of Google Assistant API service.')
@click.option('--credentials',
              metavar='<credentials>', show_default=True,
              default=os.path.join(click.get_app_dir('google-oauthlib-tool'),
                                   'credentials.json'),
              help='Path to read OAuth2 credentials.')
@click.option('--project-id',
              metavar='<project id>',
              help=('Google Developer Project ID used for registration '
                    'if --device-id is not specified'))
@click.option('--device-model-id',
              metavar='<device model id>',
              help=(('Unique device model identifier, '
                     'if not specifed, it is read from --device-config')))
@click.option('--device-id',
              metavar='<device id>',
              help=(('Unique registered device instance identifier, '
                     'if not specified, it is read from --device-config, '
                     'if no device_config found: a new device is registered '
                     'using a unique id and a new device config is saved')))
@click.option('--device-config', show_default=True,
              metavar='<device config>',
              default=os.path.join(
                  click.get_app_dir('googlesamples-assistant'),
                  'device_config.json'),
              help='Path to save and restore the device configuration')
@click.option('--lang', show_default=True,
              metavar='<language code>',
              default='en-US',
              help='Language code of the Assistant')
@click.option('--display', is_flag=True, default=False,
              help='Enable visual display of Assistant responses in HTML.')
@click.option('--verbose', '-v', is_flag=True, default=False,
              help='Verbose logging.')
@click.option('--input-audio-file', '-i',
              metavar='<input file>',
              help='Path to input audio file. '
              'If missing, uses audio capture')
@click.option('--output-audio-file', '-o',
              metavar='<output file>',
              help='Path to output audio file. '
              'If missing, uses audio playback')
@click.option('--audio-sample-rate',
              default=audio_helpers.DEFAULT_AUDIO_SAMPLE_RATE,
              metavar='<audio sample rate>', show_default=True,
              help='Audio sample rate in hertz.')
@click.option('--audio-sample-width',
              default=audio_helpers.DEFAULT_AUDIO_SAMPLE_WIDTH,
              metavar='<audio sample width>', show_default=True,
              help='Audio sample width in bytes.')
@click.option('--audio-iter-size',
              default=audio_helpers.DEFAULT_AUDIO_ITER_SIZE,
metavar='<audio iter size>', show_default=True,
              help='Size of each read during audio stream iteration in bytes.')
@click.option('--audio-block-size',
              default=audio_helpers.DEFAULT_AUDIO_DEVICE_BLOCK_SIZE,
              metavar='<audio block size>', show_default=True,
              help=('Block size in bytes for each audio device '
                    'read and write operation.'))
@click.option('--audio-flush-size',
              default=audio_helpers.DEFAULT_AUDIO_DEVICE_FLUSH_SIZE,
              metavar='<audio flush size>', show_default=True,
              help=('Size of silence data in bytes written '
                    'during flush operation'))
@click.option('--grpc-deadline', default=DEFAULT_GRPC_DEADLINE,
              metavar='<grpc deadline>', show_default=True,
              help='gRPC deadline in seconds')
@click.option('--once', default=False, is_flag=True,
              help='Force termination after a single conversation.')
def main(api_endpoint, credentials, project_id,
         device_model_id, device_id, device_config,
         lang, display, verbose,
         input_audio_file, output_audio_file,
         audio_sample_rate, audio_sample_width,
         audio_iter_size, audio_block_size, audio_flush_size,
         grpc_deadline, once, *args, **kwargs):
    """Samples for the Google Assistant API.
    Examples:
      Run the sample with microphone input and speaker output:
        $ python -m googlesamples.assistant
      Run the sample with file input and speaker output:
        $ python -m googlesamples.assistant -i <input file>
      Run the sample with file input and output:
        $ python -m googlesamples.assistant -i <input file> -o <output file>
    """
    # Setup logging.
    logging.basicConfig(level=logging.DEBUG if verbose else logging.INFO)

    # Load OAuth 2.0 credentials.
    try:
        with open(credentials, 'r') as f:
            credentials = google.oauth2.credentials.Credentials(token=None,
                                                                **json.load(f))
            http_request = google.auth.transport.requests.Request()
            credentials.refresh(http_request)
    except Exception as e:
        logging.error('Error loading credentials: %s', e)
        logging.error('Run google-oauthlib-tool to initialize '
                      'new OAuth 2.0 credentials.')
        sys.exit(-1)

    # Create an authorized gRPC channel.
    grpc_channel = google.auth.transport.grpc.secure_authorized_channel(
        credentials, http_request, api_endpoint)
    logging.info('Connecting to %s', api_endpoint)

    # Configure audio source and sink.
```

```
audio_device = None
  if input_audio_file:
      audio_source = audio_helpers.WaveSource(
          open(input_audio_file, 'rb'),
          sample_rate=audio_sample_rate,
          sample_width=audio_sample_width
      )
  else:
      audio_source = audio_device = (
          audio_device or audio_helpers.SoundDeviceStream(
              sample_rate=audio_sample_rate,
              sample_width=audio_sample_width,
              block_size=audio_block_size,
              flush_size=audio_flush_size
          )
      )
  if output_audio_file:
      audio_sink = audio_helpers.WaveSink(
          open(output_audio_file, 'wb'),
          sample_rate=audio_sample_rate,
          sample_width=audio_sample_width
      )
  else:
      audio_sink = audio_device = (
          audio_device or audio_helpers.SoundDeviceStream(
              sample_rate=audio_sample_rate,
              sample_width=audio_sample_width,
              block_size=audio_block_size,
              flush_size=audio_flush_size
          )
      )
  # Create conversation stream with the given audio source and sink.
  conversation_stream = audio_helpers.ConversationStream(
      source=audio_source,
      sink=audio_sink,
      iter_size=audio_iter_size,
      sample_width=audio_sample_width,
  )

  if not device_id or not device_model_id:
      try:
          with open(device_config) as f:
              device = json.load(f)
              device_id = device['id']
              device_model_id = device['model_id']
              logging.info("Using device model %s and device id %s",
                           device_model_id,
                           device_id)
      except Exception as e:
          logging.warning('Device config not found: %s' % e)
          logging.info('Registering device')
          if not device_model_id:
              logging.error('Option --device-model-id required '
                            'when registering a device instance.')
              sys.exit(-1)
```

```
if not project_id:
            logging.error('Option --project-id required '
                          'when registering a device instance.')
            sys.exit(-1)
        device_base_url = (
            'https://%s/v1alpha2/projects/%s/devices' % (api_endpoint,
                                                         project_id)
        )
        device_id = str(uuid.uuid1())
        payload = {
            'id': device_id,
            'model_id': device_model_id,
            'client_type': 'SDK_SERVICE'
        }
        session = google.auth.transport.requests.AuthorizedSession(
            credentials
        )
        r = session.post(device_base_url, data=json.dumps(payload))
        if r.status_code != 200:
            logging.error('Failed to register device: %s', r.text)
            sys.exit(-1)
        logging.info('Device registered: %s', device_id)
        pathlib.Path(os.path.dirname(device_config)).mkdir(exist_ok=True)
        with open(device_config, 'w') as f:
            json.dump(payload, f)

device_handler = device_helpers.DeviceRequestHandler(device_id)

@device_handler.command('action.devices.commands.OnOff')
def onoff(on):
    if on:
        logging.info('Turning device on')
    else:
        logging.info('Turning device off')

@device_handler.command('com.example.commands.BlinkLight')
def blink(speed, number):
    logging.info('Blinking device %s times.' % number)
    delay = 1
    if speed == "SLOWLY":
        delay = 2
    elif speed == "QUICKLY":
        delay = 0.5
    for i in range(int(number)):
        logging.info('Device is blinking.')
        time.sleep(delay)

with SampleAssistant(lang, device_model_id, device_id,
                     conversation_stream, display,
                     grpc_channel, grpc_deadline,
                     device_handler) as assistant:
    # If file arguments are supplied:
    # exit after the first turn of the conversation.
    if input_audio_file or output_audio_file:
```

```
 assistant.assist()
            return

        # If no file arguments supplied:
        # keep recording voice requests using the microphone
        # and playing back assistant response using the speaker.
        # When the once flag is set, don't wait for a trigger. Otherwise, wait.
        wait_for_user_trigger = not once
        while True:
            if wait_for_user_trigger:
                click.pause(info='Press Enter to send a new request...')
            continue_conversation = assistant.assist()
            # wait for user trigger if there is no follow-up turn in
            # the conversation.
            wait_for_user_trigger = not continue_conversation

            # If we only want one conversation, break.
            if once and (not continue_conversation):
                break


if __name__ == '__main__':
    main()
```

RESULT AND ANALYSIS

From this mini project,we learn the measurement of each output from each stage.The result from this output firstly from the:

.USB MIC-The mic is the most sensitive part of this project.If there is an overvoltage the Mic will burn out and start corrupting the whole Raspbian OS.We have experienced it in this project.Before we used this usb mic,we used a usb camera and used the built in mic in the camera.First,it worked good and after some days it burned out due to some over voltage problems.So, what we can say that is we have to really careful when plugging in this mic in this pi.Do not shake the USB when it is plugged in as it may corrupt the system and itself.

.SPEAKER-The speaker is also important as the mic.The mic is the input and the speaker is the output.There will be no voltage needed for this speaker as it has a built-in bactery inside.Eventhough,it has a built in bacterry,we have really careful when we plu the aux output cable,into the PI because mishandle of this aux can lead to the corrupt of the Raspbian OS.

CONCLUSION

- From this project, we learn the operation of AI machines and how this machines can ease human’s daily life.

- Beside that, understanding the concept of Phython and how to make a source code to play the Google AI Assistant.
- In future,we will upgrade our project and add more new features in our project.