Universidad de Palermo
Internet of Things

7th Noviembre 2016

Modular Smart Voice Commander

RESUMEN

Dispositivo embebido que recibe comandos de voz, se comunica con diversos servicios Cloud y ejecuta acciones de forma local.

OBJETIVOS

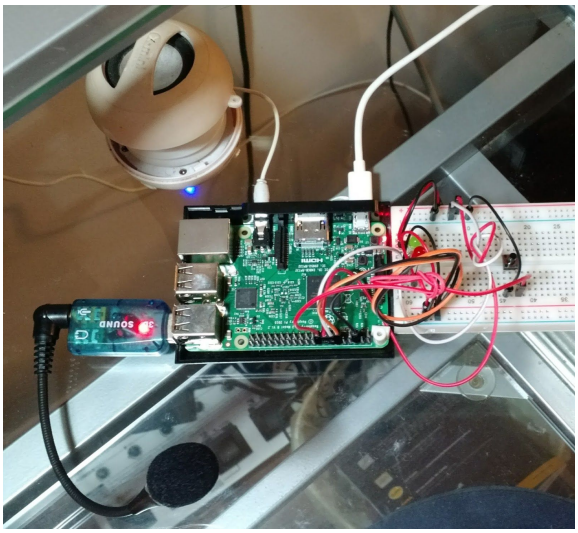
Proporcionar un sistema modular, de fácil configuración y customización dentro de una lógica de componentes intercambiables bajo una arquitectura de Internet de las Cosas y respetando las premisas del Software Libre y priorizando el uso de estándares. Que sirva para domótica, es decir para controlar de forma remota o local dispositivos hogareños, pero que que mantenga la potencialidad de ser adaptado para ser usado en otros ámbitos.

PROTOTIPO

- Raspberry Pi 3
- Raspbian OS
- Python
- Pulsador activador
- Encendido de leds vía comandos de voz
- Consultar el clima y recibir el pronóstico por voz sintetizada
- Cualquier consulta por voz al engine buscador de respuestas de Wolfram Alpha

Hernán Ordiales
h@ordia.com.ar
L: 92.881

ÍNDICE



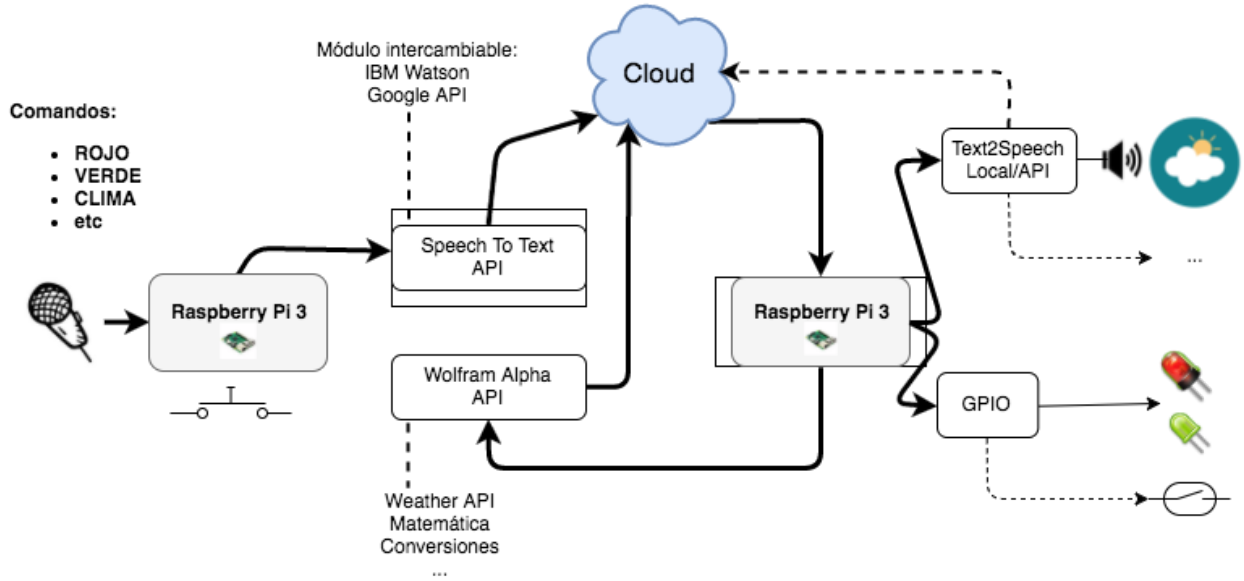
IDEA

Desarrollar un dispositivo emebebido, es decir de función dedicada, que sea capaz de responder a comandos de voz, interpretarlos y reaccionar ante los mismos. Ya sea consultando servicios externos en la nube previamente configurados, controlando o automatizando el comportamiento de artefactos hogareños como luces, webcams, persianas, parlantes, etc (domótica) y/o registrando históricos para un análisis futuro.

Entre los productos similares pensados para uso hogareño y que actualmente se encuentran en el mercado están los “hubs” Samsung SmartThings o el sistema Alexa de Amazon Echo. Todos estos se presentan como novedosos y fomentan la integración de dispositivos bajo una lógica de Internet de las Cosas.

En este proyecto lo que se busca es mantener ese concepto, pero dentro de un marco de arquitecturas abiertas y Software Libre, de forma tal que se habilite una customización y modularidad total, sin ningún tipo de limitaciones técnicas o contractuales.

ARQUITECTURA



El sistema está formado por módulos o cajas negras que respetan una interfaz, y por lo tanto, intercambiables. Los componentes a utilizar se definen en un archivo JSON (ver código). Por ejemplo para realizar el Speech To Text (conversión de voz a texto) se puede utilizar la API Cloud de IBM Watson o la de Google. Y para el procedimiento inverso, convertir el texto en voz, se puede optar por una implementación local de menor calidad Festival/Espeak o una versión Cloud de IBM.

Se apunta a que el usuario pueda customizar su producto según diversas necesidades. Por ejemplo, que pueda elegir un servicio de procesamiento Cloud o que el mismo se ejecute de forma local, según las demandas de tiempo y calidad requeridas. O seleccionar el módulo de clima o noticias de preferencia.

Módulo Entrada de Datos:

Audio: Placa de audio USB externa con driver genérico + micrófono.

Pulsadores: Pulsador común conectado a GPIO configurado con resistencia de pull-up

Módulo Speech2Text (Voz a Texto): Opciones Cloud IBM Watson API, Google API.

Módulo análisis de texto para clima y otras consultas: WOLFRAM ALPHA (cloud)

Módulo TextToSpeech: Festival/Espeak/IBM Watson (cloud)

Módulos Salida:

Módulo "Actuadores": Leds. Opcional relés.

Audio: Placa de sonido USB o integrada + Parlante

DECISIONES DE DISEÑO

Se evaluaron las diferentes opciones para construir el prototipo.

Luego de contemplar varias placas de desarrollo disponibles en el mercado local, por versatilidad, documentación y comunidad de usuarios se destacaron Arduino y Raspberry.

Se optó por la última opción debido a que incorpora de forma nativa diferentes opciones para comunicarse con internet (ethernet, wifi, bluetooth) alineándose con los objetivos del proyecto.

Del Arduino se evaluó como ventaja que posee pines "analógicos", ideales para conectar sensores directamente y por ejemplo poder discriminar valores en un sensor piezoeléctrico. Y para este caso se evaluó que la calidad de audio obtenida podía no llegar a ser suficiente.

Se concluyó que el Raspberry, aunque más caro, tiene una mejor relación calidad/precio, especificaciones de procesador, memoria, y periféricos superiores. Y mediante USB habilita a conectar cualquier placa de audio genérica que permita una resolución de 16 o más bits en la grabación. Raspberry carece de estos puertos "analógicos" pero tiene la posibilidad de

configurarle por software resistencias de pull-up y pull-down a sus pines GPIO. Algo útil por ejemplo para conectarle directamente un pulsador sin comprometer la electrónica interna.

Dispositivo	**Arduino UNO**	**Raspberry Pi 3**
Costo local ($ argentinos)	Bajo ($250)	Medio ($1000)
Procesador	Atmega 16MHz	4xARM Cortex 1.2GHz
Memoria RAM	2KB	1G DDR2
Memoria Externa	EEPROM, FLASH	SDCard
GPIO “digitales”	Si	Si (+ resistencias pull up/down)
GPIO “analógicos”	6 (10 bits de resolución)	No
Conectividad en red	Depende de módulos externos	WIFI, Ethernet, Bluetooth integrados
Bare vs OS	Firmware	Software
Audio	Pines analógicos de 10 bits para ADC u opciones externas i2c para ADC/DAC	DAC incorporado. Opciones externas via i2c o por USB para ADC

En cuanto al **sistema operativo** se evaluó windows IoT 10 y Raspbian OS (linux basado en Debian). Algunas de las cosas q se tuvieron en cuenta en la comparación

	Windows 10 IoT	**Raspbian OS**
Costo	Gratis	Gratis
Código	Privado	Software Libre y OpenSource
Ejemplos y documentación	Buena	Buena
Comunidad de usuarios	Pequeña	Muy grande
Lenguajes	C#, C++, Python, etc	C, C++, Python, etc

Se optó por Raspbian más Python como lenguaje principal, prefiriéndolos sobre la combinación WIndows 10 más C# (por ejemplo) se consideró que los primeros brindan una mayor flexibilidad a la hora de prototipar, permitiendo una integración más sencilla con aplicaciones y librerías preexistentes. Además, con Python se obtuvo la posibilidad de probar scripts de forma interactiva, además de por lo general lograr un código más simple. También se detectó que este camino contaba con una mejor comunidad y cantidad de ejemplos por

fuera de la documentación oficial. Estimando que esto podía reducir los tiempos de desarrollo. También se consideró el bug conocido que tiene Windows 10 IoT con placas de audio usb externas.

CASOS DE USO

- Presionar botón para grabar comando de voz
- Mientras graba la voz, mantener el led rojo encendido
- Si se detecta “ROJO” o “VERDE”, prender LED correspondiente
- Al preguntar por el “CLIMA”, consultar en la web el pronóstico y comunicarlo vía voz sintetizada
- Permitir la comunicación con otras API REST locales o remotas

IMPLEMENTACIÓN

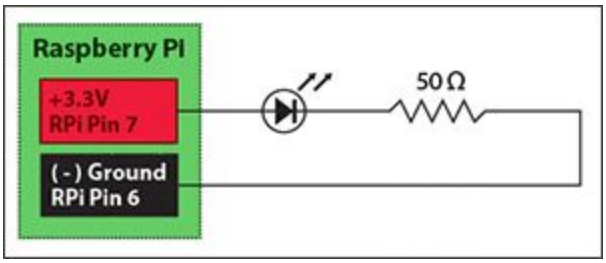
Componentes:

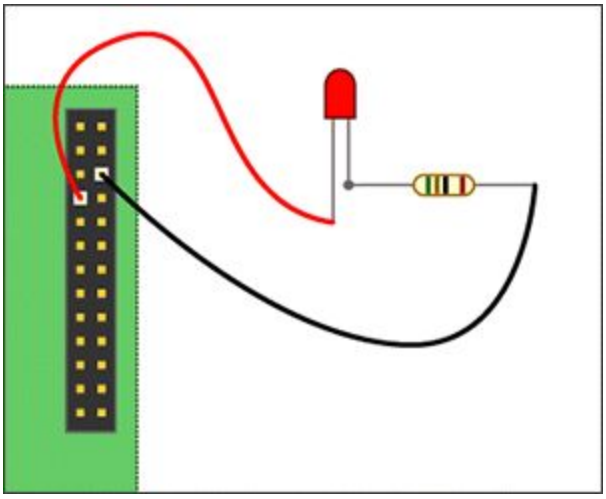
- Raspberry Pi 3 corriendo Raspbian OS en la SDCard
- Placa de audio USB genérica + micrófono
- Parlante genérico
- Push button, leds, resistencias (560 ohms) en protoboard + cables
- Conexión a internet (wifi o ethernet)

Prototipo

Conexión de Leds

Colocación de resistencia entre el circuito formado por los pines de alimentación, tierra y el led para limitar la corriente circulante (aplicación de la ley de Ohm). A menor resistencia, mayor corriente circulante y mayor brillo ($I = V/R$)

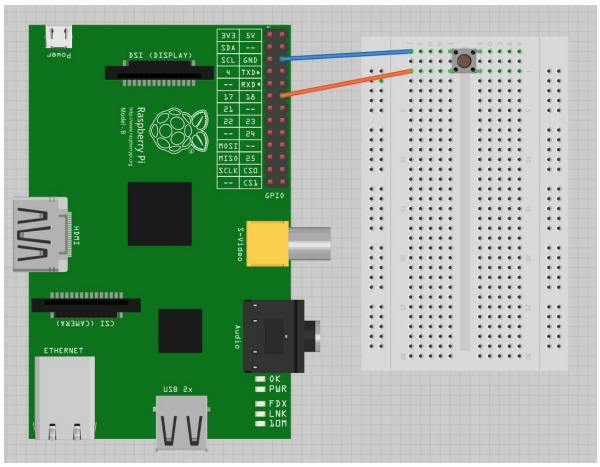




Código Python para encender un Led:

```
import RPi.GPIO as GPIO
PIN = 18
GPIO.setmode(GPIO.BCM)
GPIO.setup(PIN,GPIO.OUT)
GPIO.output(PIN,GPIO.HIGH)
print "LED encendido"
```

Conexión Push Button

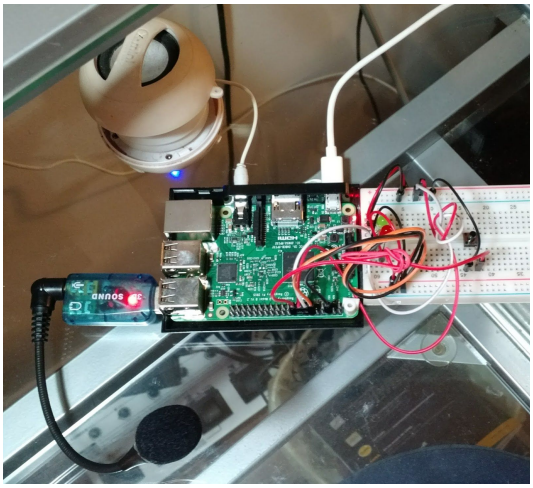


Código Python para recibir eventos:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
PIN = 24
```

```
GPIO.setup(PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
while True:
   input_state = GPIO.input(PIN)
   if input_state == False:
       print('Botón presionado')
       time.sleep(0.2)
```

Mic vía placa USB y parlante externo



APIs Cloud

Todas las APIs utilizadas usan el formato json como respuesta. Se pueden consultar vía una http query tradicional y parsear el resultado de la llamada, o utilizar sus librerías para diversos lenguajes (ver Anexo)

- IBM Watson Bluemix (demo por 30 días)
    - Speech2Text
    - Text2Speech
    - Posibilidad de agregar otros servicios orientados a IoT
- Wolfram Alpha API (buscador de respuestas general) Gratis
    - Clima
    - Conversor de unidades
    - Ejemplos: link
- Google API (demo por 60 dias)
    - Speech2Text
    - Posibilidad de agregar otros servicios orientados a IoT

Tarifas IBM Bluemix
Speech To Text:
Los primeros mil minutos son gratuitos
Text To Speech:
El primer millón de caracteres es gratuito

API Local

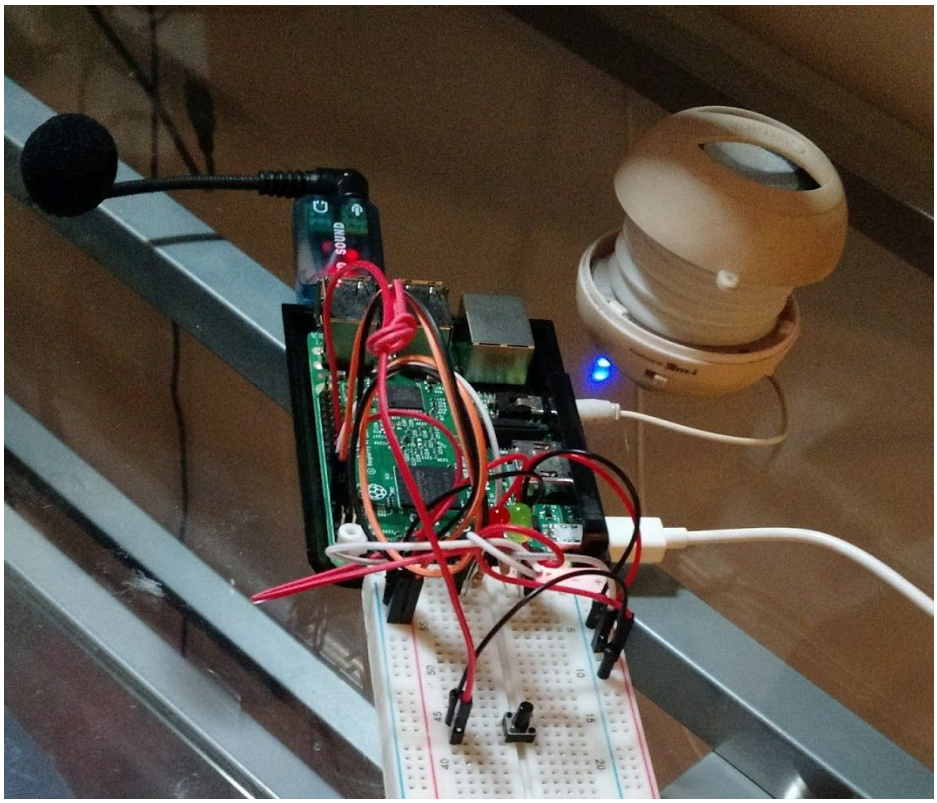
Se desarrolló un servidor REST API del tipo “mock” para hacer pruebas locales, la misma se implementó con flask (ver Anexo). Esto permitió poder hacer queries locales y recibir como respuesta archivos json, parsearlos y simular una conexión a internet y a los servicios cloud.

Problemas durante el desarrollo

- Problemas con la fuente de alimentación o puertos USB de notebooks
    - Solución: Fuentes que entreguen >0.85A
- Problemas con la calidad de la SDCard (clase, velocidad)
    - Solución: Clase 10 y buena marca
- Servicios Cloud en modo demo por pocos días (30 o 60) y luego condicionadas al uso de una tarjeta de crédito internacional
    - Solución: Suscribirse al servicio y limitarse a lo ofrecido gratuitamente, pero esto no permitió explorar otras posibilidades con las que contaban estas plataformas
- Bug en arecord al grabar en 16 bits (calidad CD)
    - Solución: Grabación en 8 bits y modo “narrow band” en la API text2speech (en lugar de “broad band”)
- Desarrollo en lugares con conexión de red de modo corporativo (usuario y contraseña) no soportadas por el Raspberry. Caso de las aulas de la universidad
    - Solución: Utilización de un router propio y la conexión compartida de un celular personal
- Detección de “lima” cuando se decia “clima” u “ojo” cuando se decia “rojo”
    - Solución: Se tomaron como casos particulares válidos

Demo

La prueba de concepto consistió en decir las palabras “ROJO”, “VERDE” y “CLIMA” y recibir como respuesta el feedback auditivo (voz sintetizada) y visual (leds)

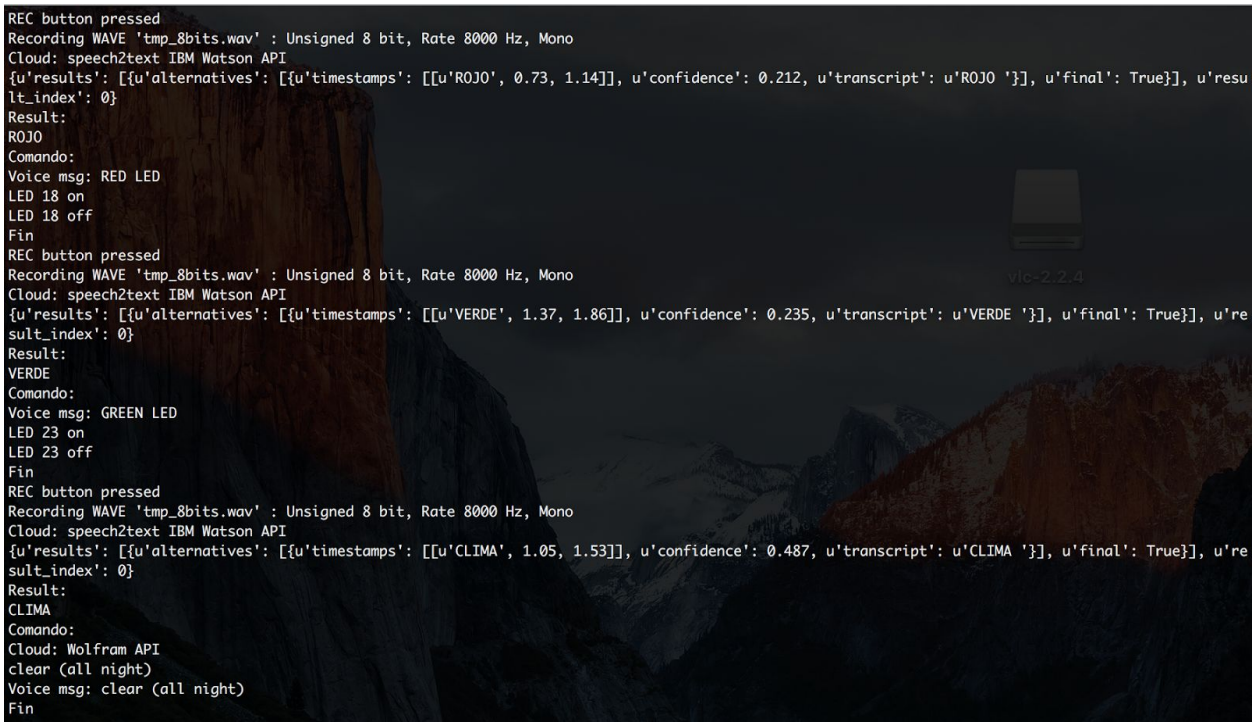


Modo de uso:

- Presionar el pulsador para empezar a hablar
- Decir claramente el comando (en castellano) cerca del micrófono mientras el led rojo se encuentra encendido
- Esperar el procesamiento del mismo (esto puede variar según el tipo de conexión a internet)
- Observar el feedback visual en leds y auditivo (pronóstico o información general en inglés) según corresponda

Link al video demostrativo: https://www.youtube.com/watch?v=DqKZwT7mEv8

Salida (log):



TRABAJO FUTURO

Principalmente poder Incorporar un histórico en Cloud y extraer conocimiento de esos datos, ya sea detectando patrones de uso y anticipándose a ellos, o generando recomendaciones al usuario. Otra posibilidad es relevar el tipo de uso efectivo del producto (tipos de comandos recibidos, horarios, etc) con fines de ajustar el marketing o las características del producto.

En cuanto a funcionalidades, por un lado está la posibilidad de mejorar las características actuales. Por ejemplo solucionar el bug de arecord (o investigar alternativas) para que permita grabar audio en 16 bits y por lo tanto utilizar el modo “Broadband” de mayor calidad en la conversión del voz a texto que proporciona IBM Watson. Investigar las posibilidades de un conversor de voz que funcione en modo local y no dependa de la red o suscripciones es otra vía, incluso imitando el modelo de funcionamiento de smartphones actuales que según posibilidades de conexión a internet y tiempos de procesamiento utilizan un modo u otro (local o cloud)

Hacer uso del parámetro de “confidence” que aportan las APIs de speech2text y descartar comandos que no superan determinado umbral.

Profundizar en las técnicas de interpretación de los comandos, por ejemplo para dar por válidos aquellos que se asemejen a los esperados, entendiendo no solo lo que el usuario dijo,

sino lo que quiso decir. Por ejemplo, en el citado caso de entender “ojo” por “rojo” o “lima” por clima”. Automatizar la detección de estos casos particulares. Para esto se pueden aprovechar las técnicas asociadas al BigData (para almacenar los datos) y el Analytics (para extraer patrones).

En cuanto a los usos asociados a la domótica, utilizar relés como actuadores para activar/desactivar todo tipo de artefactos hogareños desde lámparas a heladeras, integrar con controles remotos IR y cualquier otro tipo de tecnología que permita la automatización y control remoto. Por ejemplo desde una aplicación para smartphones y realizando la conexión con el dispositivo vía internet.

Mejorar la calidad de la voz sintetizada localmente y agregar el soporte para idioma castellano (actualmente en inglés).

Explorar las posibilidades de llevar esta tecnología a ambientes industriales, posiblemente más ruidosos y por ejemplo aplicar técnicas de preprocesamiento de audio que eliminen el ruido de ambiente (noise reduction) o permitir reconocer solo una voz preconfigurada (útil para sistemas de seguridad). Evaluar su aplicación en dispositivos que estén por fuera del ámbito hogareño, como ser funcionar como computadora de abordo de una bicicleta y utilizar la conexión a internet de un celular para mantener la parte cloud (información online del camino, recomendaciones, etc)

Construir la lógica principal como módulo para https://nodered.org y permitir al usuario final interconectar con otras APIs de forma gráfica y desde un browser, asi como programar actuadores no contemplados desde el código. Crear recetas.

CONCLUSIÓN

Se logró un prototipo funcional en los tiempos esperados sin mayores problemas, en parte debido a la extensa documentación disponible en la web y en parte a la calidad de las herramientas disponibles que actualmente permiten combinar diferentes cosas en poco tiempo.

Se pudo experimentar con las API Cloud de forma exitosa y aprovechar sus ventajas en cuanto a calidad y diversidad de resultados. Pero se notó que para aplicaciones de este tipo la demora en procesar (sobre todo en la conversión de voz a texto) puede ser excesiva creando una experiencia de usuario poco agradable.

Se vio limitada la posibilidad de explorar en mayor profundidad las herramientas que brindan las soluciones de BigData y Analytics debido a que solo se contó con suscripciones en modo demo por poco tiempo y con algunas funcionalidades restringidas. Se notó que el enfoque

actual del negocio está en cobrar por las cantidades medibles (ancho de banda, almacenamiento, etc) cloud, ya que a nivel software cliente, las opciones a nivel sistema operativo actualmente son gratuitas (Windows IoT 10 o las alternativas basadas en linux) así como sus toolchains y aplicaciones derivadas.

ANEXO A

Documentación consultada

- Página oficial de Raspberry PI https://www.raspberrypi.org
- Documentación oficial de Raspberry, Raspbian, Windows IoT 10, Python.
- Especificaciones y ejemplos proveídos por IBM Watson, Google, Wolfram Alpha para la utilización de sus APIs.
- Tutoriales en la web.

ANEXO B

Configuración

Configurar Raspbian OS en una SDCard y bootear el dispositivo con la misma.

Procesamiento local

Text to Speech

$ sudo apt-get install festival

o

$ sudo apt-get install espeak

Voice recording

5 segundos de grabación en 8 o 16 bits (narrow and broad band detection) utilizando el hardware de audio usb externo (device: 1)

8 and 16 bits 8 bits: $ arecord -d 5 -D plughw:1,0 test8.wav

16 bits: $ arecord -d 5 -f cd -D plughw:1,0 test16.wav

APIs Cloud

Rest API WebService (mock)

$ sudo pip install flask

Wolfram Alpha

API Key (free): http://products.wolframalpha.com/api/

$ sudo pip install wolframalpha

Queries: forecast and other general requests

IBM Watson

API (demo): https://developer.ibm.com/watson/

Use: Speech2Text, Voice Synthesis (text2speech)

$ sudo pip install --upgrade watson-developer-cloud

Google Cloud

API (demo): https://cloud.google.com/speech/docs/

Use: Speech2Text

$ git clone https://github.com/GoogleCloudPlatform/python-docs-samples.git

$ cd python-docs-samples/speech/api-client

$ virtualenv env

$ source env/bin/activate

$ pip install -r requirements.txt

Others

Weather: forecast.io, http://www.wunderground.com/weather/api/

Código Fuente

Repositorio online: https://github.com/hordiales/iot-modular-voice-cmd/
Licencia: GPLv3

Voice_cmd.py

```
#!/usr/bin/python2
# Author: Hernán Ordiales <hordiales@gmail.com>
import os
import subprocess
import time
import sys
import json

# Cloud APIs
from watson_developer_cloud import SpeechToTextV1
import wolframalpha

# rpy
import RPi.GPIO as GPIO

# Config file
with open('iot-config.json', 'r') as f:
  config = json.load(f)

# GPIO setup
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
PIN_RED_LED = int(config['led.red'])
PIN_GREEN_LED = int(config['led.green'])
GPIO.setup(PIN_RED_LED,GPIO.OUT)
GPIO.setup(PIN_GREEN_LED,GPIO.OUT)

PIN_REC_BTN = int(config['btn.rec'])
GPIO.setup(PIN_REC_BTN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

```
# Cloud APIs initialization
speech_to_text = SpeechToTextV1(
  username=config['watson.username'],
  password=config['watson.password'],
  x_watson_learning_opt_out=False
)
audio_type = 'audio/wav'
continuous = False # short phrases

app_id=config['wolfram.app_id']
client = wolframalpha.Client(app_id)

def turnOnOffLed(pin=PIN_RED_LED, sleeptime=1):
 print("LED %i on"%pin)
 GPIO.output(pin,GPIO.HIGH)
 time.sleep(sleeptime)
 print("LED %i off"%pin)
 GPIO.output(pin,GPIO.LOW)

def record_voice_cmd(duration,filename):
  '''
    16 bits: arecord -f cd -D plughw:1,0 test16.wav
  '''
  # arecord bug with 16 bits (many wav files)
  #rec_cmd = "arecord -d %i -f cd -D plughw:1,0 %s"%(duration,filename)
  rec_cmd = "arecord -d %i -D plughw:1,0 %s"%(duration,filename)
  subprocess.call(rec_cmd, shell=True)

seconds = 4
audio_quality = "8bits" # o 16bits
rec_filename = ""
audio_model = ""
if audio_quality=="8bits":
  rec_filename = "tmp_8bits.wav"
  audio_model = 'es-ES_NarrowbandModel'
else:
  rec_filename = "tmp_16bits.wav"
  audio_model = 'es-ES_BroadbandModel'

def voice(txt):
  subprocess.call("echo \"%s\" | festival --tts "%txt, shell=True)
```

```
def detect_and_cmd():
 GPIO.output(PIN_RED_LED,GPIO.HIGH) # on
 record_voice_cmd(seconds,rec_filename)
 GPIO.output(PIN_RED_LED,GPIO.LOW) # off

 with open(join(dirname("__file__"), rec_filename), 'rb') as audio_file:
        #Watson API
        print("Cloud: speech2text IBM Watson API")
        a = json.dumps( speech_to_text.recognize(
              audio_file, content_type=audio_type, timestamps=True, model=audio_model, continuous=continuous) )
        #Google API

        b = json.loads( a )
        print( b )
        print( "Result: ")
        try:
          print( b['results'][0]['alternatives'][0]['transcript'] )
          transcript = b['results'][0]['alternatives'][0]['transcript']
          transcript = transcript.strip()
          transcript = transcript.lower()
          #TODO: check confidence value
        except Exception, e:
          print(e)
          transcript = ""
        print("Comando: ")
        if "ojo" in transcript: # ROJO 'ojo'/'rojo
            voice("RED LED")
            turnOnOffLed(pin=PIN_RED_LED, sleeptime=2)
        elif "erde" in transcript: #VERDE 'erde'/'verde''
            voice("GREEN LED")
            turnOnOffLed(pin=PIN_GREEN_LED, sleeptime=2)
        elif "lima" in transcript: # Acepta 'clima', 'lima' o 'el clima'
            print("Cloud: Wolfram API")
            query="Forecast Buenos Aires"
            res = client.query(query)
            output = next(res.results).text
            temp, state = output.split('\n')
            print(state)
            clima = state
            voice(clima)
        else:
            print("No se detecto comando")
```

```
            voice(transcript)


 print("End")
#detect_and_cmd()

#TODO: run as startup service https://www.raspberrypi.org/documentation/linux/usage/rc-local.md
if __name__ == '__main__':
 while True:
   #WARNING: cloud API consumption
   input_state = GPIO.input(PIN_REC_BTN)
   if input_state == False:
      print('REC button pressed')
      detect_and_cmd()
      time.sleep(0.2)
```

Write-config-file.py

```
import json

watson_config = {'watson.username': '', 'watson.password': ''}

wolframalpha_config = {'wolfram.app_id': ''}

leds_config = {'led.red': '18', 'led.green': '23'}

# spech2text: watson / google / local? / LAN_Rest_API

# text2speech: festival / espeak

services_config = {'speech2text': 'watson', 'text2speech': 'festival', 'record': 'arecord'}

# quality (and broad or narrowband model): 8bits vs 16bits (bug with 16bits and arecord)

audio_qualitiy = {'audio.quality': '8bits'}

def merge_dicts(*dict_args):
  result = {}
  for dictionary in dict_args:
     result.update(dictionary)
  return result


config = merge_dicts(watson_config, wolframalpha_config, leds_config, services_config, buttons_config)


print(config)

with open('iot-config.json', 'w') as f:
  json.dump(config, f)
```