Universidad Autónoma de Querétaro
Facultad de Informática
Ingeniería en Computación

Domótica utilizando un asistente virtual implementado en la tarjeta Raspberry Pi.

Opción de titulación
Tesis

Que como parte de los requisitos para obtener el Título de Ingeniero en Computación.

Presenta:
Luis Alberto Martínez Guapo

Dirigido por:
Dr. Jesús Carlos Pedraza Ortega

Dr. Jesús Carlos Pedraza Ortega Presidente	Firma
M.C. Ricardo Chaparro Sánchez Secretario	Firma
Dra. Ma. Teresa García Ramírez Vocal	Firma
Dr. Fausto Abraham Jacques García Suplente	Firma
M. en C. Jesús Martín Jaramillo Morales Suplente	Firma

M.I.S.D Juan Salvador Hernández Valerio
Director de la Facultad de Informática

Centro Universitario
Querétaro, Qro.
Octubre 2018
México

Domótica utilizando un asistente virtual
implementado en la tarjeta Raspberry Pi.

por

Luis Alberto Martínez Guapo

Clave RI: IFLIN-230897

RESUMEN

Se presenta la aplicación de asistentes virtuales en domótica, siendo el desarrollo de este prototipo un desarrollo completamente *OpenSource*. Se implementó un asistente virtual con interacción por comandos de voz en una tarjeta Raspberry Pi 3 haciendo uso de los servicios de voz a texto (STT) y texto a voz(TTS) que proporciona Google Cloud Platform, utilizando lenguaje Python. Se implementó y se incorporó un protocolo UART para comunicación serial entre el asistente virtual (tarjeta Raspberry Pi) y una tarjeta de desarrollo Arduino, para el control de actuadores y el monitoreo de sensores de interés.

Palabras clave: Asistente Virtual, Voz a Texto(STT), Texto a Voz (TTS), Motor Lógico (Logic Engine), Automatización de tareas, Edificios Inteligentes.

Dedicatoria

“Nuestra vida es una travesía en este mundo, somos el resultado de experiencias, lecciones, conocimiento y personas. Personas que sin importar el tiempo que estén en nuestras vidas, dejan un pedacito de ellas en nosotros, y de igual forma dejamos un pedacito de nosotros en ellas. Pedacitos de personas que nos hacen reflexionar, cambiar, mejorar y crecer. A esas personas, gracias por complementar nuestra existencia”.

Luis Martínez, 2018.

Dedicado a mis padres: María Luisa y Alberto, que siempre me han apoyado y confiado en mí, ¡gracias!

AGRADECIMIENTOS

Gracias a Dios, por permitirme llegar hasta aquí. A mis padres y mi familia, que siempre me han apoyado, que han confiado en mí, que me han escuchado y aconsejado.

A mis amigos y compañeros de la carrera: a Said, Sebastián, Diego, Chaps, Juanjo, Henry, Carapia, Mayra, Miguel, a Cesar por prestarme su Laptop para terminar este trabajo. Gracias por los buenos momentos que pasamos, por el apoyo mutuo que hubo durante toda la carrera, por su amistad.

A los amigos que conocí durante la carrera: a Dany, Andy, Jona, que, aunque eran de otras carreras y generaciones, me brindaron su amistad y apoyo. A Sara con quien compartí agradables momentos y de quién también aprendí, gracias.

A mi amiga Ale, que siempre está para escucharme y aconsejarme.

Al profesor Ricardo Chaparro que me ayudo durante toda la carrera con becas, conocimiento y consejos, Gracias profe. A mi amigo y tocayo Luis Alberto, de quien aprendí desarrollo web y que también me ha brindado mucha ayuda, gracias Luis.

Al Dr. Juan Manuel Ramos Arreguín que me proporcionó un espacio en la Facultad de Ingeniería para terminar este trabajo. Al Dr. Jesús Carlos Pedraza Ortega, quien es mi asesor y que me ha brindado todo su apoyo para llevar a cabo este proyecto, por los consejos y pláticas que vuelven agradable el ambiente de trabajo, gracias.

Debo agradecer también a mis amigos y ex compañeros de trabajo de Everis, con quien pasé muchos agradables momentos, con quienes aprendí y crecí profesionalmente, quienes me brindaron su amistad y hacían que los momentos difíciles y estresantes no fueran tan malos, algo que me permitió avanzar poco a poco en este proyecto. A Abel, Edwin, Mike, Soler, Zavala, Gabo, JC, a Gaby. A Tere Salas, Armando, José Luis y Osciel por el apoyo, confianza y consejos que me brindaron, los cuales me han ayudado a mejorar profesionalmente y como persona, y a Gus por darme la oportunidad de ingresar a la empresa.

TABLA DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE DIAGRAMAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

El presente trabajo incursiona en el novedoso mundo de los asistentes virtuales con interacción por comandos de voz con el propósito de darles aplicación en el campo de la domótica, para ello se hace uso de una tarjeta embebida Raspberry Pi, en la cual se ejecuta el asistente virtual.

Se hace uso de una placa Arduino para realizar las tareas de automatización esto para que el sistema sea modular y poder reducir el procesamiento en la tarjeta Raspberry Pi, y que la Raspberry destine recursos a otras tareas.

Se propone un sistema domótico basado en un asistente virtual, y con un desarrollo completamente *OpenSource*. Se muestran todas las etapas de desarrollo del asistente y su integración en la domótica, además de un vistazo al amplio potencial que estos asistentes virtuales nos brindan.

Se proponen, además, nuevas posibles aplicaciones para los asistentes virtuales en diferentes entornos, personalizados con ciertas características para cumplir objetivos específicos y de esta forma brindar mayor confort y seguridad a los usuarios.

1.1 Objetivos

1.1.1 Objetivo general

Desarrollar un asistente virtual con reconocimiento de voz implementado en la tarjeta Raspberry Pi utilizando librerías *OpenSource*, y mediante interfaces con tarjetas basadas en microcontroladores, se puedan ejecutar tareas de automatización simples indicadas por el usuario.

1.1.2 Objetivos específicos:

- Desarrollar un asistente virtual capaz de reconocer comandos de voz en una tarjeta Raspberry Pi utilizando librerías *OpenSource*.
- Adaptar el asistente virtual para que sea capaz de utilizar los pines digitales de entrada y salida de la tarjeta Raspberry Pi.
- Diseñar e implementar una interfaz de comunicación entre el asistente virtual (Raspberry) con alguna tarjeta basada en microcontrolador.

- Hacer pruebas de distintos sensores y actuadores en alguna tarjeta basada en microcontrolador.
- Realizar pruebas del sistema completamente integrado y analizar los resultados.

1.2 Justificación

El desarrollo de este proyecto es para explotar el potencial tecnológico que nos brinda la inteligencia artificial en conjunto con otras áreas del conocimiento como lo es el software embebido y las interfaces de hardware, con las cuales se busca generar sinergia, innovación y generación de conocimiento.

Este asistente virtual, se pretende desarrollar como un sistema embebido, el cual pueda tener interacción con dispositivos de hardware como lo son las tarjetas de desarrollo basadas en microcontroladores, los cuales, a su vez, puedan activar sensores, ¿y por qué no?, que sea capaz de interactuar con varios de estos dispositivos. La activación o interacción con dispositivos de hardware se pretende realizar a través de comandos simples de voz.

1.3 Descripción del problema

Los asistentes virtuales que existen en el mercado se encuentran disponibles en su mayoría para computadoras, dispositivos móviles y/o usables (wearables), algunos alojados como servicios en la nube (cloud). En la mayoría de los casos, su funcionamiento está limitado por las empresas que los fabricaron. Estos asistentes interactúan por medio de lenguaje natural, al recibir instrucciones por voz y responden de la misma manera.

Al día de hoy el producto más parecido a lo que se propone en este trabajo, es Echo Dot de Amazon, el cual hace uso del asistente virtual Alexa, propio de Amazon, aunque el concepto bajo el que está concebido es *SmartHouse*. Este producto es capaz de interactuar con dispositivos de algunos fabricantes tales como LG, Samsung entre otras.

Este tipo de desarrollos tecnológicos, empieza a ser más y más común entre usuarios, pero aún no lo es tanto entre desarrolladores, lo cual

se vuelve un área de oportunidad en cuanto a explotación tecnológica para ampliar las aplicaciones que se le pueden dar a un asistente virtual, y a la inteligencia artificial en particular.

1.4 Planteamiento teórico

Es factible el desarrollo e implementación de un asistente virtual con librerías *OpenSource* en una tarjeta embebida de bajo costo, con la finalidad de poder automatizar tareas simples del hogar y lograr un ahorro en el costo de este tipo de desarrollos tecnológicos.

Los asistentes virtuales no son tan comunes, sin embargo, son muy novedosos y muy interesantes por la capacidad y cantidad de desarrollo de aplicaciones que se les puede dar. Si bien, esto da paso a lo que es una casa inteligente. Esto podría, además, ser la base para incorporarle al asistente virtual un sistema de seguridad, y todo esto, alojado como un sistema embebido en una tarjeta Raspberry Pi.

En términos de equipo necesario para su desarrollo es económico, y el consumo de energía sería bastante bajo, lo que lo vuelve muy rentable.

1.5 Alcances y limitaciones

1.5.1 Alcances

- Propuesta de aplicación de los asistentes virtuales en el área de la domótica.
- Desarrollar un prototipo de sistema domótico operado a través de un asistente virtual con interacción por comandos de voz.
- Sistema desarrollado bajo el sistema operativo Raspbian.
- Realizar pruebas de funcionalidad y usabilidad del prototipo.

1.5.2 Limitaciones

Las librerías que se utilizan requieren acceso a internet, y para obtener excelente performance en tiempos de respuesta se requiere de una conexión a internet de al menos 10 Mbps/s para tener un óptimo rendimiento.

Las librerías de procesamiento de voz sólo permiten 60 minutos de procesamiento al mes de forma gratuita por IP. Para uso mensual, de 61 a 1,000,000 minutos, se tiene un costo de 0,006 UDS por cada 15 segundos de audio procesado.

El micrófono adquirido tiene poco alcance.

1.6 Organización de la tesis

En el capítulo 1 se describe el proyecto de investigación, se exponen los objetivos, el problema de investigación, la justificación, los alcances y las limitaciones.

El capítulo 2 presenta el marco teórico, el cual permite entrar en un contexto más técnico para el entendimiento de diferentes conceptos utilizados en el desarrollo del proyecto. Además, permitirá conocer las áreas tecnológicas que abarca el proyecto.

Por su parte, el capítulo 3 expone a detalle cada una de las etapas que se llevaron a cabo para el desarrollo del proyecto, también se hace referencia a las herramientas utilizadas.

Los resultados del proyecto son presentados en el capítulo 4, se exponen los resultados de pruebas unitarias, pruebas de integración, pruebas posteriores a las correcciones y resultados finales. Se presentan también el rendimiento y la usabilidad del sistema.

Para concluir, en el capítulo 5 se expresan las aportaciones del proyecto, así como el trabajo a futuro que supone mejoras y factibilidad de escalamiento o expansión.

2. MARCO TEÓRICO

2.1 Asistentes virtuales

La llegada de los asistentes virtuales ha tenido un importante evento en la historia de la computación. Los asistentes virtuales son muy útiles ayudando a usuarios de sistemas computacionales automatizando y realizando tareas con la mínima interacción hombre-máquina.

La interacción que se da entre un asistente virtual y una persona debe ser natural, una persona se comunica usando la voz y el asistente virtual lo procesa e interpreta y responde de la misma manera (Pant, 2016)

2.1.1 Arquitectura de un asistente virtual

El software primordial consiste en tres componentes principales, componente de voz a texto (STT por sus siglas en inglés), el motor lógico (*Logic Engine*) y el componente de texto a voz (TTS por sus siglas en inglés) como se observa en la Figura 1.

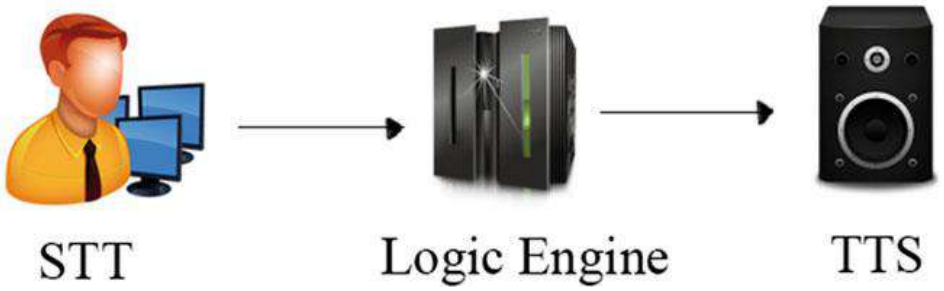

Figura 1. Flujo de trabajo de un asistente virtual (T. Pant. 2016).

El **STT** convierte las instrucciones de voz del usuario a una cadena de texto que pueda ser procesada por el motor lógico. Esto involucra grabar la voz del usuario, capturar las palabras desde la grabación y entonces, usar un procesador de lenguaje natural para convertir la grabación a una cadena de texto.

El **Motor Lógico** es el componente de software que recibe la cadena de texto desde el STT, procesa la entrada y pasa la salida al TTS. El motor lógico puede ser considerado el cerebro del asistente virtual, maneja las consultas del usuario por medio de una serie de cláusulas programadas. El

asistente decide cual salida debería darse de respuesta para una entrada específica.

El **TTS** es el componente que recibe la salida del motor lógico y convierte la cadena de texto a instrucciones de voz para completar la interacción con el usuario. Es una parte de suma importancia, pues hace al asistente virtual más humano.

Estos componentes del sistema dejan fuera cualquier tipo de interacción física entre el usuario y la computadora, los usuarios pueden interactuar con el sistema de la misma manera en que interactúan con otras personas (Pant, 2016).

2.2 Edificios inteligentes

El concepto de edificio inteligente surge de la sinergia de diferentes disciplinas como la informática, la microelectrónica, las telecomunicaciones, la arquitectura y la automática. Este concepto se comenzó a utilizar en foros informáticos para hacer referencia a sistemas capaces de procesar datos y conseguir un comportamiento similar al humano incorporando al edificio inteligencia artificial para simplificar el mantenimiento (Romero, Vázquez & Castro, 2011) algunas de las características más importantes se muestran en el Diagrama 1.

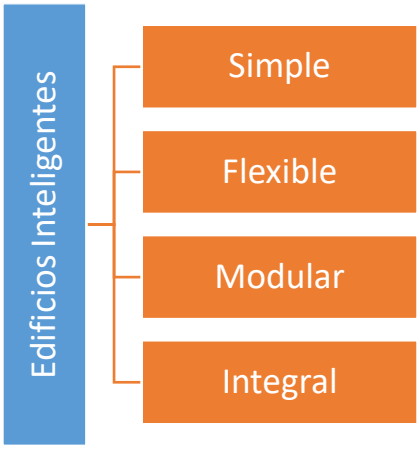

Diagrama 1. Características de un edificio inteligente.

Podemos definir edificio inteligente a aquel edificio que ha sido dotado con cierto grado de autonomía (automatización, tecnología, inteligencia artificial) para realizar tareas por sí mismo.

La clasificación más habitual de los sistemas a gestionar es aquélla que los agrupa dependiendo del tipo de servicio como se presenta en el Diagrama 2:

Diagrama 2. Servicios habituales a gestionar.

2.2.1 Edificio domótico

Para cuestiones de automatización del hogar tenemos la domótica, que se aplica a la ciencia y a los elementos desarrollados por ella que proporcionan algún nivel de automatización o automatismo dentro de la casa. La vivienda domótica es aquella que integra una serie de automatismos en materia de electricidad, electrónica, robótica, informática y telecomunicaciones, con el objetivo de asegurar al usuario un aumento del confort, seguridad, ahorro energético, facilidades de comunicación, y de las posibilidades de entretenimiento. Busca la integración de todos los aparatos del hogar (Huidobro & Millán, 2010).

Para el desarrollo de este trabajo, se trabajará bajo el concepto de domótica.

2.2.2 Edificio inmótico

La inmótica se entiende como la incorporación de sistemas de gestión técnica automatizada a las instalaciones del sector terciario como son plantas industriales, hoteles, hospitales, aeropuertos, edificios de oficinas, parques tecnológicos, grandes superficies, universidades, instalaciones comunitarias en edificios de viviendas, etc. Un sistema inmótico realiza el control permanente de cada uno de los elementos que intervienen en un edificio (Huidobro & Millán, 2010).

2.3 Sistemas embebidos

Se puede definir los sistemas embebidos como un sistema electrónico de procesamiento programado para realizar funciones o tareas para un objetivo específico. Cuando los sistemas son más grandes se pueden incluir además de elementos electrónicos y de software, partes mecánicas, eléctricas y electromecánicas (Aceves & Ramos, 2012).

Los sistemas embebidos están presentes en nuestro día a día. La sociedad ignora en gran medida la cantidad de sistemas embebidos con los que interactúan diariamente. Como lo son teléfonos celulares, computadoras, tablets, pantallas inteligentes, entre otros.

El software embebido difiere del software convencional en una computadora por las siguientes características:

- Los sistemas embebidos pueden ser en tiempo real.
- Tienen una interfaz directa con el hardware del dispositivo.
- Se consideran las pruebas de hardware como parte del desarrollo del sistema embebido.
- Hay más control del tiempo de respuesta y manejo de recursos como memoria.
- Requiere de recursos sumamente especializados en las áreas informáticas, electrónicas, etc.

Desde su aparición, estos sistemas se han vuelto indispensables en la mayoría de los dispositivos electrónicos. La industria electrónica cambia y crece, de manera que necesita de modelos y arquitecturas de diseño más complejas y completas en cortos ciclos de desarrollo. Los dispositivos programables lograron cambiar los esquemas de desarrollo al integrar hardware y software en componentes reconfigurables, conservando atributos vitales como eficiencia, costo, tiempo de desarrollo, entre otros (Aceves & Ramos, 2012).

2.3.1 Sistemas basados en dispositivos reconfigurables

Para el desarrollo de sistemas computacionales se han planteado dos posibilidades bien definidas. Por un lado, la posibilidad de "ejecución en

software", donde la tarea se realiza mediante la realización de un conjunto de instrucciones. Se trata de una solución muy flexible porque permite ejecutar diferentes tareas, aunque la eficiencia es menor dada la funcionalidad general del microprocesador. Por otro lado, la posibilidad de "ejecución en hardware", donde se desarrolla un circuito específico para la tarea a realizar. Esta solución presenta un mejor rendimiento dada la especialidad del diseño y el posible paralelismo de la aplicación, pero reduce la flexibilidad del sistema debido a la imposibilidad de cambiar la funcionalidad una vez fabricado.

Los dispositivos reconfigurables, se presentan como una solución intermedia entre el uso de microprocesadores de propósito general y el diseño de circuitos específicos (ASIC). En una situación ideal los sistemas reconfigurables combinan lo mejor de las dos soluciones anteriores: la flexibilidad del software que se ejecuta en un microprocesador de propósito general y la velocidad del hardware específico (González M, 2006). Dentro de estos dispositivos se tiene, por ejemplo: los PLDs, CPLDs y FPGAs.

2.3.2 Sistemas basados en microprocesadores

La tecnología de procesadores trata de la arquitectura del núcleo computacional usado para implementar las funcionalidades deseadas de un sistema. Un microprocesador es un componente LSI (*Large Scale Integration*) que realiza una gran cantidad de funciones o tareas en una sola pieza de circuito integrado (Pérez A., 2009).

En cierta medida el software que se suele instalar en una PC es software embebido, software embebido basado en microprocesador. Dispositivos como las tarjetas Raspberry Pi y Beaglebone, son basadas en microprocesadores.

2.3.2.1 Raspberry Pi

Raspberry Pi es una computadora del tamaño de una tarjeta de crédito diseñada originalmente para la educación, inspirada en el 1981 BBC Micro. El objetivo del creador Eben Upton era crear un dispositivo de bajo costo que mejorara habilidades de programación y comprensión de hardware en el nivel medio superior. Pero gracias a su pequeño tamaño y el precio accesible, fue rápidamente adoptado por los fabricantes de herramientas y

entusiastas de la electrónica para proyectos que requieren más que un microcontrolador básico (*UpSkill Learning*, 2016).

La Raspberry Pi es más lenta que una computadora portátil o computadora de escritorio moderna, pero sigue siendo una computadora Linux completa y puede proporcionar todas las habilidades esperadas que implica, con un bajo nivel de consumo de energía.

Se opta por utilizar Raspberry por la amplia documentación y librerías que existen actualmente, además de que el precio de la tarjeta es muy accesible de acuerdo con las características que tiene en comparación con otras tarjetas con características similares como lo es la Beaglebone Black. En la figura 2 se muestran las tarjetas ya mencionadas, y en la Tabla 1 la comparativa de características.

Figura 2. Tarjetas de desarrollo basadas en microprocesador, Beaglebone Black Rev C y Raspberry Pi 3 Model B. (Amazon, 2017).

Tabla 1. Características de tarjetas basadas en microprocesador.

	Beaglebone Black Rev C	Raspberry Pi 3 Model B
Procesador	Arm Cortex-A8 de 1 GHz	1.2GHz 64-bit quad-core ARMv8 CPU
Memoria RAM	Memoria RAM de 512 MB DDR3	1 GB RAM
USB	1 x USB 2.0	4 x USB 2.0

Video	Interfaz HDMI tipo D	Full HDMI
GPU	sgx530	VideoCore IV 3D graphics core
Precio	$1,294.96	$1,149.00

Si bien la diferencia en el precio mostrada en la Tabla 1 es muy poca, sí difieren en la capacidad de procesamiento, pero en cuanto a documentación, antecedentes y librerías, Raspberry termina por ser mejor opción (Amazon, 2017).

2.3.3 Sistemas basados en microcontroladores

Un microcontrolador es un circuito integrado de propósito específico, y está conformado por componentes similares al de una computadora de escritorio tales como CPU, memoria, etc., pero no incluye ningún dispositivo de comunicación con humanos (monitor, teclado, etc.).

Es también un microprocesador ASIP (*Application Specific Instruction set Processor*) que ha sido optimizado para aplicaciones de control embebidas. Estas aplicaciones monitorean y fijan numerosas señales de control de un único bit, pero no realizan cálculos exhaustivos (Pérez A., 2009). Se tienen algunos ejemplos de microcontroladores comerciales y de bajo costo como son los PIC(Microchip) y Atmega(Atmel).

2.3.3.1 Arduino

Arduino es una plataforma de electrónica de código abierto basado en hardware y software fácil de usar. Precisamente la tarjeta Arduino que aparece en la Figura 3, tiene como cerebro a un microcontrolador Atmel, el ATmega328… Las tarjetas Arduino pueden leer entradas (provenientes de sensores) y generar salidas para activar dispositivos (actuadores), para esto sólo basta con enviar un conjunto de instrucciones al microcontrolador de la tarjeta. Para hacerlo, se utiliza el lenguaje de programación de Arduino (que es básicamente lenguaje C++), cuyo entorno de desarrollo está basado en Processing (Arduino, 2017).

El entorno de programación de Arduino (Processing), es muy amigable y muy intuitivo, además viene precargado con gran variedad de ejemplos que ayudan al usuario a aprender a utilizar estas tarjetas con mayor facilidad y rapidez para el desarrollo de proyectos.

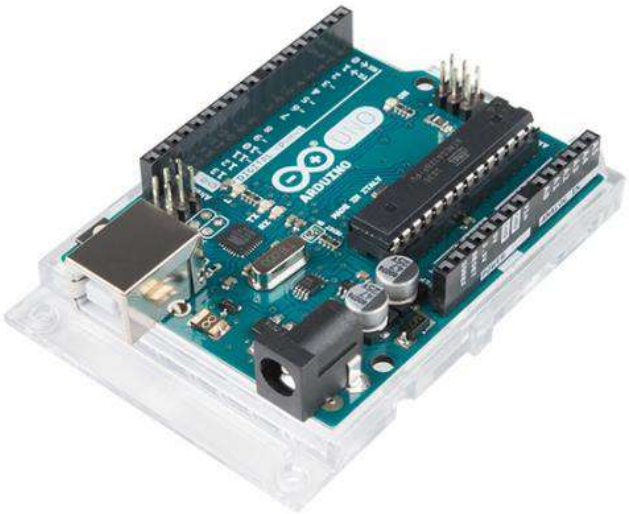

Figura 3. Placa Arduino Uno (Arduino, 2017).

Arduino es una muy eficiente plataforma para desarrollar prototipos, que es lo que se busca. Se elige Arduino por la simplicidad que brinda en el desarrollo de proyectos, además de la gran cantidad de documentación, librerías y antecedentes con los que cuenta esta tarjeta, siendo esta una de sus principales fortalezas y ventajas sobre otras tarjetas de desarrollo basadas en microcontroladores. Cabe mencionar que es una tarjeta muy accesible en cuanto a su costo, pues la tarjeta más básica (Figura 3) la podemos adquirir desde $430.00 MXN.

2.3.3.2 Sensores

Para comprender lo que es un sensor es necesario entender lo que es un transductor y su funcionamiento. Un transductor se define como aquel dispositivo que es capaz de convertir una variable física en otra que tiene un dominio diferente (Corona, Abarca & Mares, 2014). De acuerdo con lo anterior, podemos deducir que un transductor forma parte de un sensor.

Un sensor es un transductor que cambia el dominio de una variable física y que además proporciona una salida útil para ser usada como variable

de entrada en un sistema de procesamiento de información (un microcontrolador, por ejemplo).

El uso de sensores es una parte básica en el desarrollo de sistemas domóticos, pues monitorean el estado de ciertas variables físicas dentro del edificio, variables que comprometen el confort y la seguridad del usuario.

2.3.3.3 Actuadores

Los transductores también forman parte de los actuadores, estos se encargan de ejecutar la acción determinada por el sistema de procesamiento de la información. Por ende, de manera general, se dice que un transductor cambia la variable física medida en un movimiento, en presión, en flujo, en una señal eléctrica, etc. (Corona, Abarca & Mares, 2014).

Los actuadores también son parte básica en el desarrollo de sistemas domóticos, más en general, de sistemas de automatización, pues son estos dispositivos los que generan el cambio de estado de algún tipo sobre un elemento. Los actuadores son los encargados de automatizar las tareas que en un principio eran realizadas por el usuario.

3. METODOLOGÍA

Previo al desarrollo del proyecto, se realizó un análisis de requerimientos de hardware para identificar los dispositivos y componentes que se requerían investigar y adquirir como parte de la infraestructura del prototipo. Se realizó también un análisis de requerimientos de software para definir las funcionalidades que debían ser implementadas.

Con base a los análisis de requerimientos realizados, se estableció una metodología de desarrollo modular con etapas de integración y pruebas, esto para reducir de manera considerable la dependencia entre las actividades planificadas.

La metodología diseñada se conforma de 8 fases, las cuales se muestran gráficamente en el Diagrama 3:

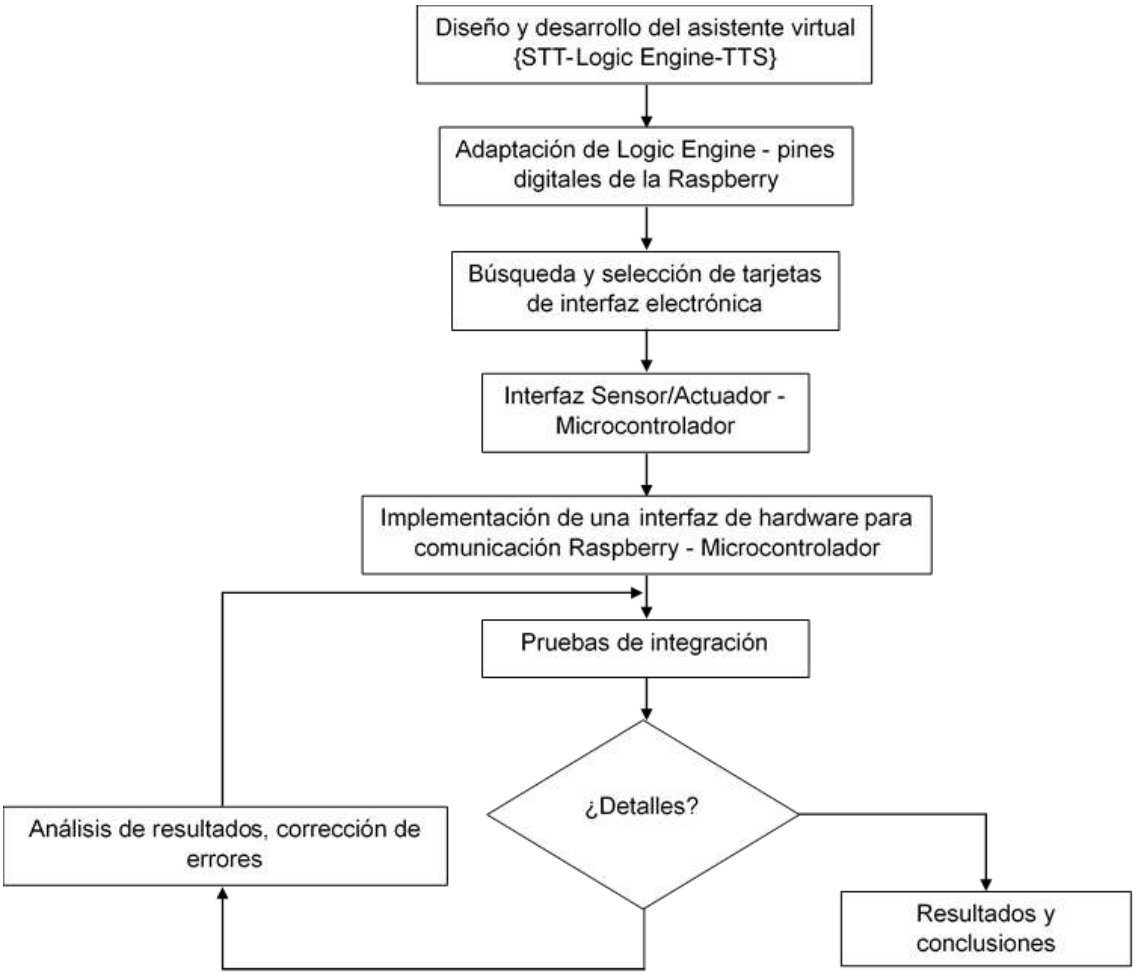

Diagrama 3. Metodología.

3.1 Diseño y desarrollo del asistente virtual

Como ya se mencionó en el capítulo 2, un asistente virtual está conformado de tres elementos básicos, Voz a Texto, Motor Lógico y Texto a Voz. Uno de los propósitos de este proyecto es ampliar los alcances del asistente virtual convencional, por lo que el diseño técnico que se propone es el que se muestra en la Figura 4:

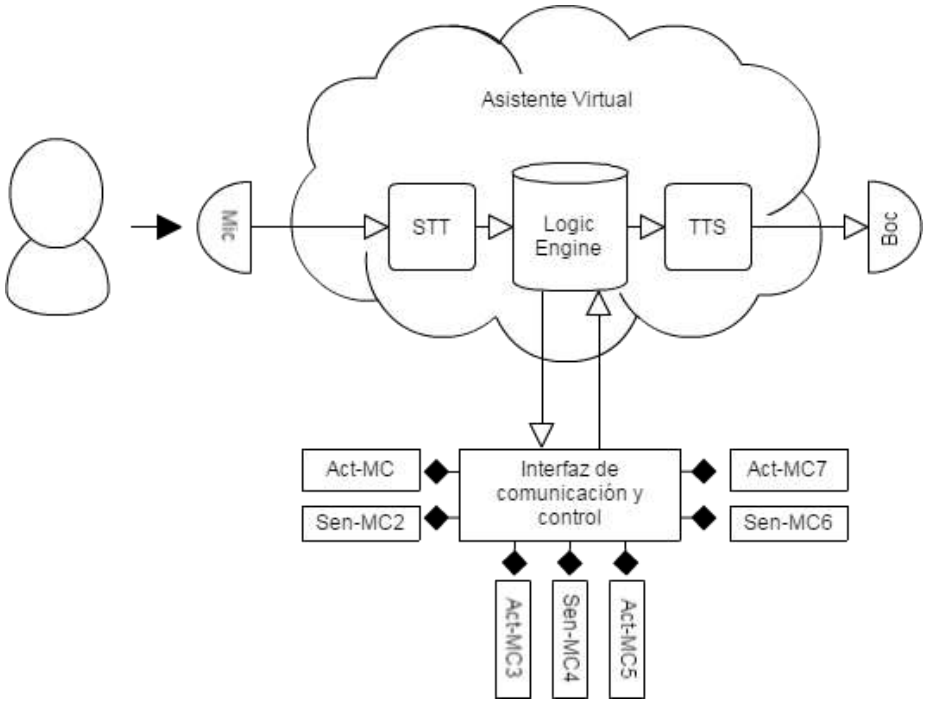

Figura 4. Flujo de operación del sistema.

A continuación, se describen los componentes del diseño técnico propuesto:

- Mic: Micrófono, este es nuestro receptor de comandos de voz.
- Voz a Texto (STT): este es el componente (Uso de API) que nos permitirá procesar el comando de voz y convertirlo a texto.
- Logic Engine: Este componente es el core del sistema (toma de decisiones).
- Texto a Voz (TTS): este es el componente (Uso de API) que sintetizará texto a voz.
- Boc: Bocina o dispositivo de salida de audio.

Los componentes ya mencionados conforman únicamente al asistente virtual. Los componentes siguientes son los encargados de la automatización de tareas:

- Interfaz de comunicación y control: Este es el medio por el cual el asistente virtual indicará las tareas que se deben ejecutar a los microcontroladores.
- MCX: Microcontrolador, dispositivo encargado de interactuar con los sensores y actuadores, de acuerdo con las instrucciones recibidas por el asistente virtual. Este tipo de componente será cubierto por placas Arduino.
- Act: Actuador, dispositivo que ejecutara las instrucciones relacionadas a tareas del hogar.
- Sen: Sensor, dispositivo encargado de monitorear variables físicas de importancia en el hogar.

Se consideró utilizar lenguaje Python para el desarrollo del proyecto, dado que es un leguaje muy potente, que brinda mucha facilidad en la manipulación de API's, la sintaxis es muy amigable y permite su fácil integración en sistemas embebidos, adecuado para lograr los objetivos del proyecto.

3.1.1 Estructura del proyecto

Es necesario definir una estructura para el proyecto, de manera que el código se mantenga ordenado. La estructura es la siguiente:

```
VirtualAssistant/
    main.py
    LogicEngine/
      __init__.py
      brain.py
      AssistantSkills/
          __init__.py
          *.py
          SpeechTextEngines/
              __init__.py
              stt.py
              tts.py
```

Donde:

- VirtualAssistant: es la carpeta que contendrá el proyecto

- SpeechTextEngines: Este será el módulo donde se alojarán los componentes SpeechToText y TextToSpeech del proyecto.
- LogicEngine: Este módulo contendrá los componentes que fungirán como el cerebro del asistente virtual (toma de decisiones).
- AssistantSkills: Este módulo contendrá los componentes en los cuales se implementan las funcionalidades del asistente virtual (funcionalidades de un asistente convencional y de automatización).
- __init__.py: Este archivo es la forma en que Python inicializa paquetes o módulos, de esta manera indicamos que los directorios que contienen este archivo son módulos y nos permitirán importar las librerías que se encuentran en el módulo.
- main.py: Este será el componente principal del proyecto, el que ejecutará el asistente virtual.

3.1.2 Motor de Voz a Texto (STT)

Para este componente se hace uso del API Google Speech Recognition, la cual tiene soporte en diferentes lenguajes y a la cual podemos acceder instalando el paquete SpeechRecognition por medio de comando pip.

Para realizar la conversión de voz a texto sólo es necesario invocar el método “recognize_google” que proporciona el paquete speech_recognition, el cual debe ser parametrizado con el lenguaje y una grabación, una que podemos grabar utilizando el método “listen” que nos proporciona el mismo paquete, sólo es necesario parametrizarlo con una referencia al micrófono, un timeout y un tiempo límite de la frase. Si el servicio es capaz de procesar la grabación devolverá una cadena de texto con la frase que se indica en la grabación, de otra manera ocurrirá un error el cual se controla manejando una excepción.

Esta lógica quedará implementada en el componente stt.py en un único método que definido como stt(), el cual retornará una cadena de texto.

3.1.3 Motor de Texto a Voz (TTS)

Para este componente se hace uso del API Google TTS (gTTS), la cual también tiene soporte para diferentes lenguajes, a la cual podemos acceder instalando previamente el paquete gTTS por medio de comando pip.

Esta API contiene el método “gTTS” el cual se parametriza con una cadena de texto, y el lenguaje en el que se quiere sintetizar el audio que este método devolverá y el cual se guardará en una variable “tts”, en lenguaje Python no existen tipos de variables, posteriormente se guardará este audio en la ubicación en la que se indique invocando al método save del objeto tts: tts.save(“tts.mp3”).

Una vez que la cadena de texto ya fue sintetizada a audio y este a su vez ya fue guardado físicamente en memoria, y haciendo uso de la librería “os”, se invoca el siguiente método para reproducir el audio guardado: os.system(“mpg321 tts.mp3”), donde mpg321 es un reproductor multimedia para sistemas basados en Linux, el cual se instala con el comando “apt-get –y install mpg321”. Al ejecutar el comando system, se reproducirá el audio que se generó. Esto quedará implementado en el método tts() en el componente tts.py.

3.1.4 Motor Lógico (Logic Engine)

Para esta etapa se define la forma en que el asistente virtual realiza la toma de decisiones. En este componente se validarán: la cadena de texto que devuelve el método “stt” y el mensaje o código emitido por algún microcontrolador (Se detallará en el apartado 3.5) para validarlo y tomar una decisión.

De manera general se toma una cadena de texto o frase, se pasa la cadena a minúsculas y por último se parte la cadena en un arreglo de palabras con ayuda de la función “split “. Haciendo uso de instrucciones “if” se valida si alguna de las palabras obtenidas de la cadena de texto, se encuentra en el arreglo de palabras conocidas contra las que se está comparando. Validado esto, se invoca el método que ejecutará la funcionalidad solicitada, la cual puede ser funcionalidad del asistente convencional o funcionalidad de automatización.

El Motor Lógico tomará decisiones por tres flujos diferentes. El primer flujo es el del asistente virtual convencional que se presenta en la Figura 5.

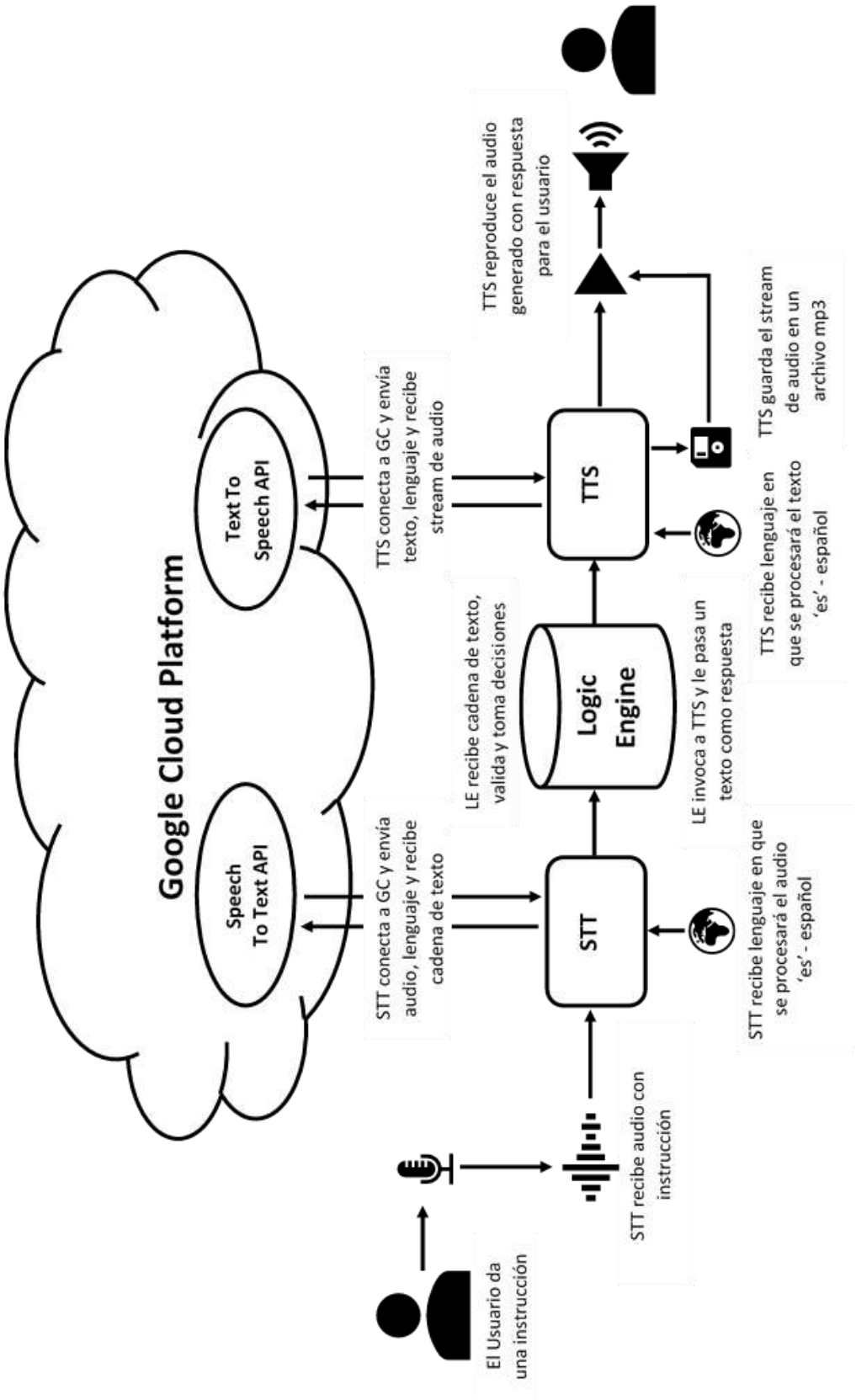

Figura 5. Flujo de operación detallado del asistente virtual convencional.

Un usuario pregunta algo al asistente virtual, y este sintetiza un audio con la respuesta para el usuario.

En esta etapa, es posible validar tantas instrucciones como skills sean implementados. Existen una gran cantidad de API's que van desde consultar el clima, hora, fecha hasta crear una lista de reproducción de música. Cabe mencionar que existen API's de carácter educativo, como lo es la API de Wikipedia, que permite consultar gran variedad de temas, los cuales no tendremos que leer, sino que el mismo asistente virtual lo redactará para nosotros. También cabe la posibilidad de permitir que el asistente virtual tenga acceso a una base de datos de tipo enciclopedia, esto también para fines educativos.

A partir de esta etapa, para ampliar las características sólo bastará con implementar la funcionalidad de interés y validar alguna o algunas palabras clave para invocar dicha funcionalidad. Estas validaciones quedarán en el componente brain.py, en el que se invocarán las funcionalidades implementadas.

3.2 Adaptando el Motor Lógico – Pines digitales de la Raspberry

En esta etapa se comprueba la factibilidad del uso de asistentes virtuales en la domótica, para esto se realiza una simple prueba que consiste en dar al asistente virtual la instrucción "luces" para que el STT la procese y devuelva una cadena de texto con el mismo valor ("luces") Figura 6.

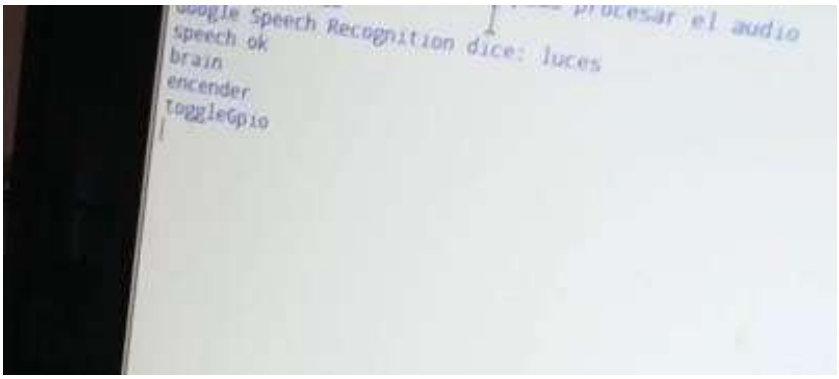

Figura 6. Reconocimiento de la instrucción "luces".

Para el control de los pines digitales se hace uso de un ensamblador de Raspberry a protoboard y de la librería GPIO (*General Purpose Input Output*) librería de entradas y salidas de propósito general. Sólo se necesita

importar la librería con “import RPi.GPIO as GPIO“, configurar GPIO a través de su método setmode y pasando como parámetro “GPIO.BOARD” para indicar que nos dirigiremos al pin por su número en la placa. Como siguiente paso se deberán configurar los pines que se desean utilizar ya sea como una salida o una entrada. A través del método “setup” y parametrizando con el número del pin y su indicador de entrada (GPIO.IN) o salida (GPIO.OUT).

Una vez que la instrucción se valida, se procede a activar uno de los pines digitales de la Raspberry enviando una señal en alto para encender un led como se ve en la Figura 7. Esto invocando al método “output” del objeto GPIO y parametrizando con el número de pin que se desea controlar y el estado en que se desea poner (True o False).

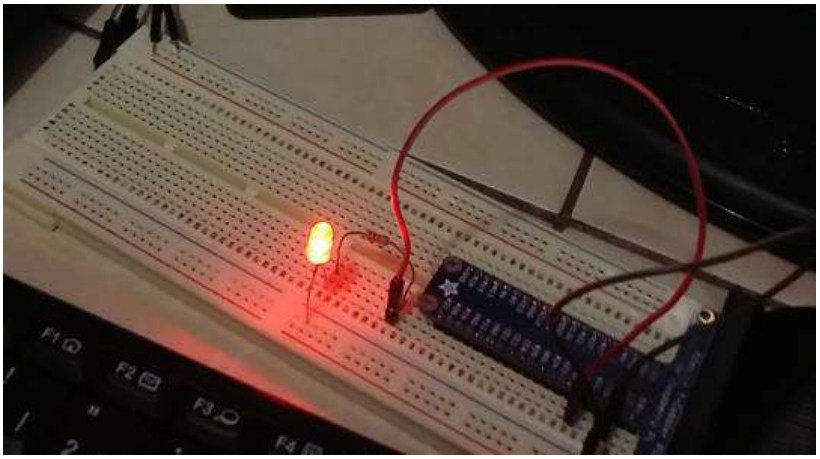

Figura 7. Encendido de led al validar la instrucción.

Para apagar el led sólo se es necesario dictar la misma instrucción que para encenderlo como aparece en la Figura 8, pues en el programa se declara una variable asociada al estatus del led, el cual se estará alternando entre encendido (True) o apagado (False) al validar la misma instrucción.

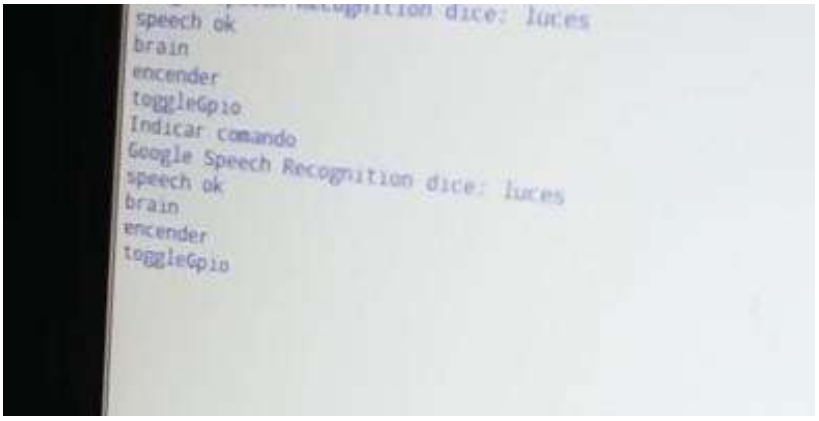

Figura 8. Se valida nuevamente la instrucción “luces”.

Una vez que se ha reconocido la instrucción “luces” nuevamente, se ejecuta la instrucción para apagar el led como se ve en la Figura 9.

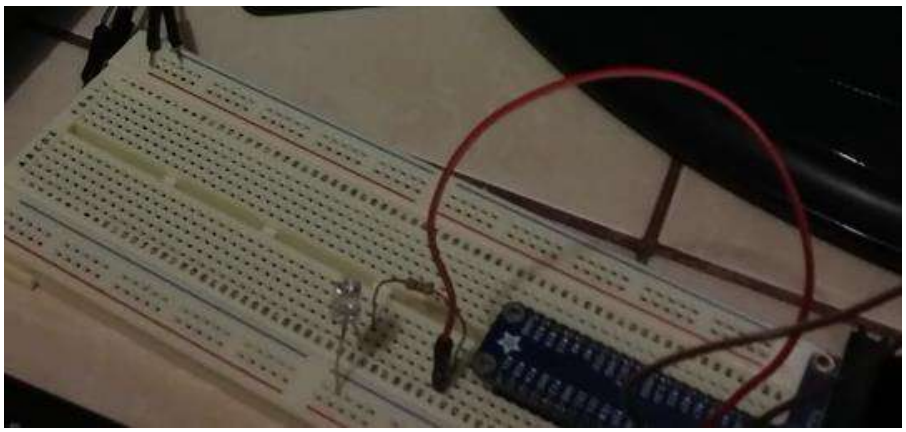

Figura 9. Apagado de led tras validar instrucción.

Tras la sencilla prueba que se realizó, se confirma la factibilidad del uso de asistentes virtuales para tareas de automatización, pues para llevar a cabo dichas tareas se requiere básicamente del control de señales digitales.

Llegado a este punto se ha implementado la mitad del flujo de automatización del asistente virtual, esto porque no es viable que el asistente virtual o, mejor dicho, la tarjeta Raspberry ejecute todas las tareas, pues a nivel de hardware es un gran desgaste ya que implica bastante procesamiento, validación de instrucciones, uso de API’s y control de señales digitales. Lo correcto es delegar el control de señales digitales a un microcontrolador (Arduino), que es un dispositivo diseñado para este tipo de tareas (Se detallará más en los apartados 3.4 y 3.5).

3.3 Tarjetas de interfaz electrónica

Al trabajar con software embebido e involucrar domótica, es muy difícil no involucrar a la electrónica sea a pequeña o gran escala. Si bien la electrónica es interesante, algo complicada y para evitar tener problemas con esta se ha tomado la decisión de involucrarla lo menos posible, por lo que se opta por la adquisición de tarjetas y dispositivos plug and play, pues esto facilita el desarrollo del proyecto, reduce los tiempos de implementación, y reduce las probabilidades de fallas. De esta manera el prototipo se vuelve más seguro y confiable a la hora de realizar pruebas unitarias y de integración.

Con este esquema el proyecto sigue siendo escalable, pues se podrán integrar más dispositivos y sólo bastará con alimentarlos, conectarlos a algún microcontrolador y agregar al mismo la lógica de programación para el dispositivo que se pretende integrar.

3.3.1 Arduino Uno

Entre los dispositivos adquiridos está la placa Arduino Uno, la cual sólo requiere alimentación eléctrica a través de un cable USB-Serial o con una fuente externa. Esta placa se encargará de ejecutar las tareas de automatización. Las características de esta tarjeta se muestran en la Tabla 2.

Tabla 2. Características Arduino Uno.

Características
Microcontrolador ATmega328.
Voltaje de entrada 7 – 12V.
Voltaje de trabajo 5V.
Corriente por pin I/O 20mA.
Corriente para pin a 3.3V 50mA
14 pines digitales I/O.
6 canales PWM.
6 ADC.
16 MHz frecuencia de reloj.
Memoria Flash 32 KB (ATmega328) de los cuales 0.5KB son usados por el bootloader.
Memoria SRAM 2KB (ATmega328).
Memoria EEPROM 1KB(ATmega328).

Información obtenida del sitio oficial de Arduino. Arduino Uno. (2017).

3.3.2 Arduino Leonardo

Se cuenta también con una placa Arduino Leonardo que se muestra en la Figura 10, la cual será la encargada de monitorear sensores y reportar alarmas al asistente virtual. La placa Arduino Leonardo que parece en la

Figura 10, se puede alimentar con una fuente externa o con un cable USB-micro USB.

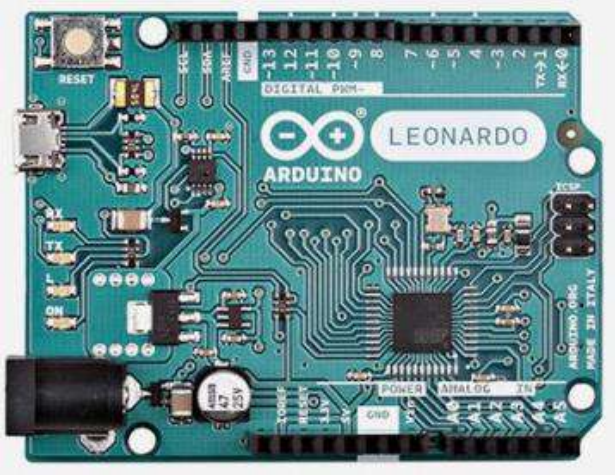

Figura 10. Tarjeta Arduino Leonardo.

Ambas tarjetas son muy similares estéticamente, pero difieren en cuanto a sus especificaciones técnicas. Las características de la tarjeta Leonardo se detallan en la Tabla 3.

Tabla 3. Características Arduino Leonardo.

Características
Microcontrolador ATmega32u4.
Voltaje de entrada 7 – 12V.
Voltaje de trabajo 5V.
Corriente por pin I/O 40mA.
20 pines digitales I/O.
7 canales PWM.
12 ADC.
16 MHz frecuencia de reloj.
Memoria Flash 32 KB (ATmega32u4) de los cuales 4KB son usados por el bootloader.
Memoria SRAM 2.5KB (ATmega32u4).
Memoria EEPROM 1KB(ATmega32u4).

Información obtenida del sitio oficial de Arduino (Arduino Leonardo, 2017. Consultado el día 19 de febrero de 2018).

3.3.3 Roles de las placas Arduino

Una de las diferencias entre estas placas es que la placa Leonardo cuenta con comunicación USB integrada y la placa Uno no, por lo que requiere un convertidor USB a Serial. Otra diferencia importante es que la placa Uno maneja eventos en comunicación serie, y la placa Leonardo no, por lo que se opta por la placa Uno para las tareas de automatización pues el programa se interrumpe para atender los eventos del puerto serie y así puede ejecutar dichas tareas a la brevedad. Por otro lado, la placa Leonardo no tiene esta característica, por lo que se utilizará para el monitoreo de sensores únicamente.

3.3.4 Módulo de relevadores (Relay)

Se adquirió también un módulo de cuatro relevadores a 5V, el que aparece en la Figura 11, para las pruebas de control de dispositivos que funcionan con corriente alterna como lo son los focos. Los relevadores de este módulo serán activados mediante la placa Arduino Uno cuando esta reciba una instrucción del asistente virtual.

Figura 11. Relay-Module 4. Inteligencia Artificial. (2017).

3.4 Interfaz Sensor/Actuador – Microcontrolador

De sensores y actuadores hay una gran variedad, los hay analógicos y digitales, en este apartado se expone la forma de usar algunos de estos

dispositivos, y la forma en que estos se pueden integrar al prototipo que se está buscando desarrollar.

3.4.1 RHT03 Sensor de temperatura y humedad

Como primera interfaz a mostrar se tiene el sensor RHT03, un sensor digital para medir temperatura y humedad. Este sensor se conectará a una placa Arduino, algunas de las características de este sensor se muestran en la Tabla 4.

Tabla 4. Características RHT03.

Modelo	**RHT03**
Fuente de alimentación	3.3 – 6V DC
Señal de salida	Señal digital a través de un bus de una línea
Elemento de detección	Polímero de condensador de humedad
Rango de operación	Humedad 0-100%RH Temperatura -40°~80° Centígrados
Exactitud	Humedad +-2%RH (Max +- 5%RH); Temperatura +-0.5° Centígrados
Resolución o sensibilidad	Humedad 0.1%RH; Temperatura 0.1° Centígrados
Repetibilidad	Humedad +-1%RH; Temperatura +-0.2° Centígrados

Para poder utilizar este sensor, hay dos opciones, la primera es implementar un protocolo de comunicación, el cual es proporcionado y explicado por el fabricante en la hoja de datos del sensor (Electrónica Embajadores, 2017). La segunda opción es utilizar la librería “DHT22.h”, la cual es software *OpenSource* (Adams Ben, 2011). Esta librería tiene implementado el protocolo de comunicación y sólo debe añadirse a las librerías del IDE Arduino.

Teniendo la librería instalada, se procede a implementar el programa para leer datos del sensor, para lo cual se utilizarán las siguientes instrucciones de código:

```
#include <DHT.h> //Se importa librería
#define PIN 5 //Se declara constante con el número de pin para lectura
#define SENSOR DHT22 //Constante con el tipo de sensor

DHT dht(PIN, SENSOR); //Se declara una instancia de la clase DHT con el número de pin a utilizar y el modelo del sensor
dht.begin(); //Se inicializa el protocolo para leer datos del sensor
int hum = dht.readHumidity(); //Se declara variable y se lee el valor de la humedad
int temp = dht.readTemperature(); //Se declara una variable y se le asigna el valor de la temperatura
int tempF = dht.readTemperature(true); //Se declara una variable y se le asigna el valor de la temperatura en grados Fahrenheit
```

Se puede dejar un tiempo de retraso entre ciclos de lectura de los datos del sensor utilizando la instrucción delay(*time*), parametrizando el tiempo que se quiere para el retraso en milisegundos. La librería nativamente devuelve valores flotantes, pero al guardarlos en variables enteras el punto flotante se pierde, si se requirieran los valores exactos bastaría con guardarlos en variables tipo float.

En la Figura 12 se muestra una implementación del código con intervalos de lectura de 1 segundo.

```
sketch_feb19a
#include <DHT.h>

#define PIN 5
#define SENSOR DHT22

DHT dht(PIN, SENSOR);

void setup() {
  Serial.begin(9600);
  dht.begin();

}

void loop() {
  delay(1000);
  int hum = dht.readHumidity();
  int temp = dht.readTemperature();
  int tempF = dht.readTemperature(true);

  Serial.print("Humedad: ");
  Serial.print(hum);
  Serial.println(" %");
  Serial.print("Temperatura: ");
  Serial.print(temp);
  Serial.print("°C ");
  Serial.print(tempF);
  Serial.println("°F");
}
```

Figura 12. Código con intervalos de lectura de 1 segundo.

El esquema de conexión (Naylamp Mechatronics, 2016) para el sensor es el que se presenta en la Figura 13.

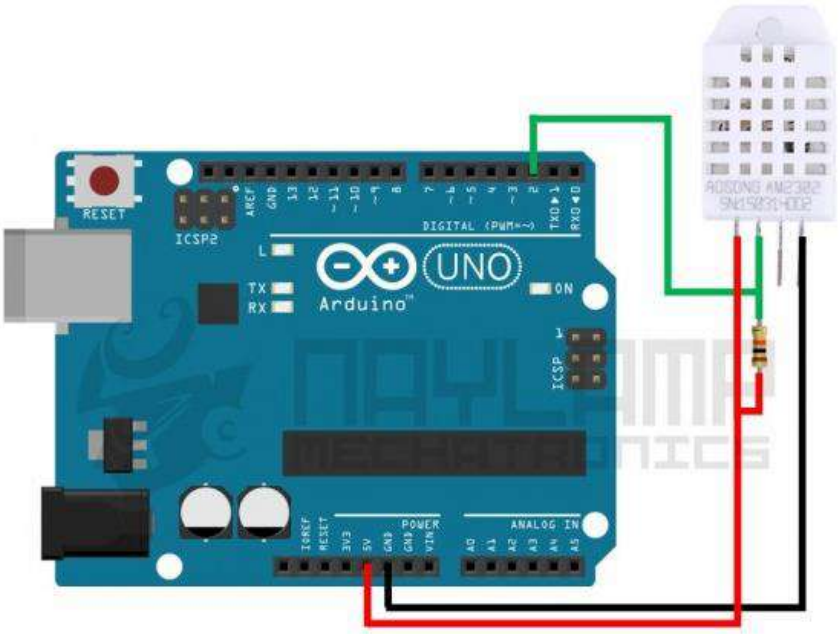

Figura 13. Esquema de conexión RHT03-Arduino.

El pin 1 del sensor a 5V, el pin 2 a una resistencia Pull-Up y una línea (a modo de divisor de voltaje) al pin de lectura de la placa Arduino, el pin 3 queda sin conectar y el pin 4 a tierra (GND). En la Figura 14 se presenta el sensor conectado a la placa Arduino.

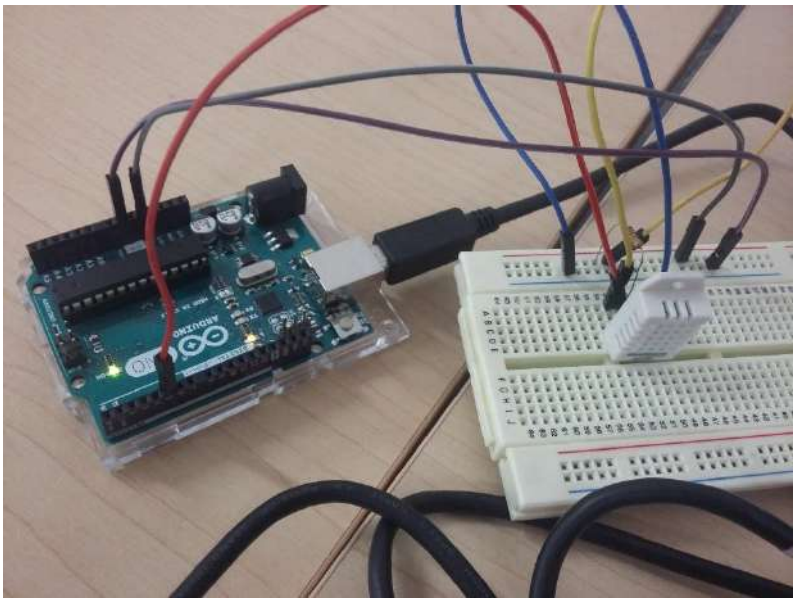

Figura 14. Circuito en funcionamiento.

La respuesta del sensor con lecturas en intervalos de 1 segundo en la placa Arduino se muestran en la Figura 15.

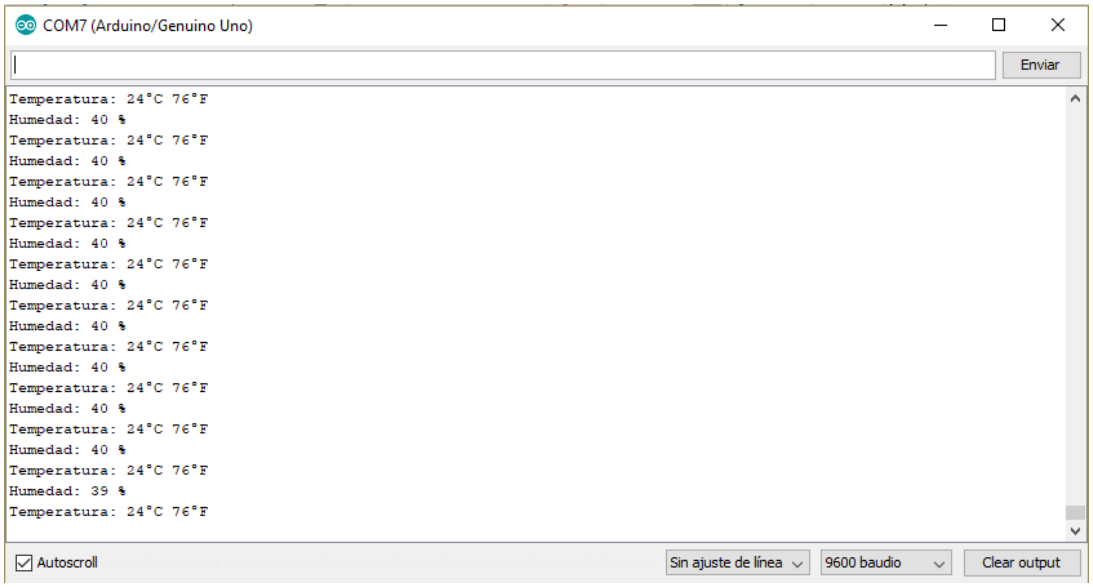

Figura 15. Respuesta del sensor RHT03 en el monitor serie del IDE Arduino.

La mayoría de los sensores digitales hacen uso de un protocolo de comunicación, por medio del cual es posible obtener las mediciones que este realiza. Estos protocolos la mayoría de las ocasiones son creados por el

fabricante, o simplemente se basan sobre algún protocolo ya existente y lo adaptan a sus necesidades.

3.4.2 Sharp GP2Y0A41SK0F - Sensor analógico de distancia

Un sensor analógico difiere un poco en comparación a un sensor digital, pues este no ocupa de un protocolo de comunicación para poder obtener las mediciones que realiza. Estos sensores devuelven un voltaje analógico dentro de un rango, en algunas ocasiones proporcional al rango en que pueden medir una magnitud física, aunque también podría ser necesario realizar algún cálculo matemático para obtener el valor real de acuerdo con lo que este dispositivo está sensando.

Como siguiente interfaz se tiene un sensor para medir distancias, cuyas especificaciones aparecen en la Tabla 5.

Tabla 5. Especificaciones Sharp GP2Y0A41SK0F.

Rango de medición	**4cm – 30cm**
Tasa de muestreo	60Hz
Fuente de alimentación	4.5V – 5.5V
Corriente de alimentación	12mA
Tipo de salida	Voltaje analógico
Voltaje diferencial de salida	2.3V

El sensor Sharp GP2Y0A41SK0F que se muestra en la Figura 16, emite una señal de luz infrarroja, la cual es reflejada por algún objeto y captada por un detector sensible a la posición (que se encuentra en el mismo sensor). El sensor devuelve un voltaje de salida correspondiente a la distancia detectada.

Figura 16. Sensor GP2Y0A41SK0F.

El valor de salida proporcionado por este sensor es no lineal, y es confiable para interpretar en términos de distancia (cm) sólo en un rango específico, en el rango indicado por el fabricante (4cm a 40cm). En la Figura 17 se muestra la relación voltaje distancia de este sensor.

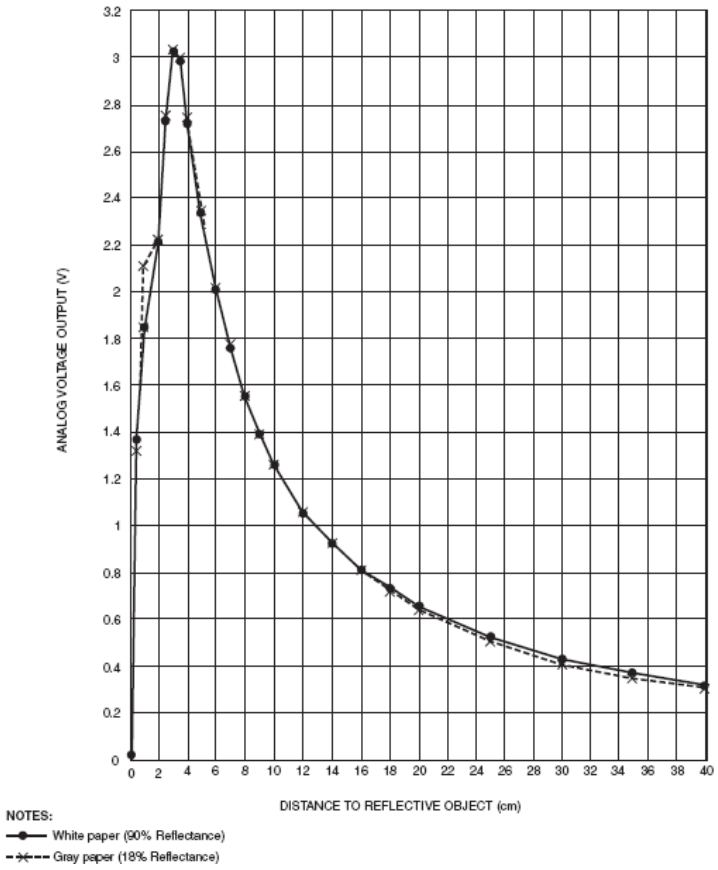

Figura 17. Representación gráfica de la respuesta del sensor. (Pololu Robotics & Electronics, 2018).

Para poder interpretar el valor analógico que devuelve el sensor, se emplea un método de interpolación cúbica. Se toman los puntos (voltaje, distancia) comprendidos en el rango (2.72V,4cm) – (0.93V,14cm) y con ayuda de la plataforma en línea WolframAlpha (WolframAlpha, 2018) se obtiene la ecuación polinómica de 3er grado para el rango de puntos ya mencionados. Se toma también el rango de puntos (0.93V,14cm) – (0.31V,40cm) y se realiza una segunda interpolación para tener mejor precisión en las mediciones del sensor. Se toman dos interpolaciones porque WolframAlpha sólo permite la interpolación de máximo 10 puntos y el total de puntos que se quieren interpolar son 16.

La ecuación polinómica para el rango de 14cm a 40cm es:

$$-68.6117x^3 + 190.245x^2 - 193.295x + 84.4158$$

La ecuación polinómica para el rango de 4cm a 14cm es:

$$-1.80101x^3 + 12.7382x^2 - 32.7002x + 34.9159$$

Dónde x es el voltaje analógico que lee la tarjeta Arduino proveniente del sensor GP2Y0A41SK0F.

En las Figuras 18 y 19 se muestra una implementación del código con intervalos de lectura de 2 segundos utilizando interpolación cubica. Si el sensor devuelve un voltaje mayor o igual a 0.31V y menor a 0.93V se utilizará la ecuación para el rango de 14cm a 40cm. Si el sensor devolviera un voltaje mayor o igual a 0.93V y menor o igual a 2.72V se utilizará la ecuación para el rango de 4cm a 14cm.

SensorDistancia

```
int sensorPin = A0;
float analogValue = 0;
float sensorValue = 0;
float distance = 0;
float coeffsH[] = {84.4158,-193.295,190.245,-68.6117}; //de 14 a 40
float coeffsL[] = {34.9159, -32.7002, 12.7382, -1.80101}; //de 4 a 14

float getDistance(){
  distance = 0;
  if(sensorValue >= 0.31 && sensorValue<0.93){
    distance += coeffsH[3] * pow(sensorValue,3);
    distance += coeffsH[2] * pow(sensorValue,2);
    distance += coeffsH[1] * sensorValue;
    distance += coeffsH[0];
  }else if(sensorValue >=0.93 && sensorValue<=2.72){
    distance += coeffsL[3] * pow(sensorValue,3);
    distance += coeffsL[2] * pow(sensorValue,2);
    distance += coeffsL[1] * sensorValue;
    distance += coeffsL[0];
  }
  return distance;
}
```

Figura 18. Codificación de las ecuaciones de interpolación.

```
void setup() {
 Serial.begin(9600);
}

void loop() {
  analogValue = analogRead(sensorPin);
  sensorValue =  analogValue*(5.0/1023);

  Serial.print("Analógico: ");
  Serial.println(analogValue);
  Serial.print("Voltaje: ");
  Serial.println(sensorValue);
  Serial.print("Distancia: ");
  Serial.print(getDistance());
  Serial.println("cm");
  delay(2000);

}
```

Figura 19. Lectura de valores del sensor e interpretación en voltaje.

El esquema de conexión para el sensor de distancia se muestra en la Figura 20.

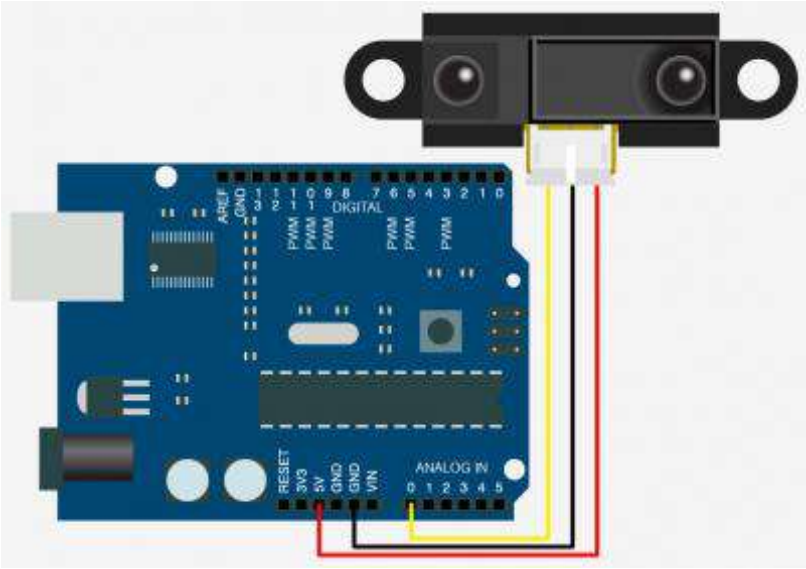

Figura 20. Esquema de conexión Sharp GP2Y0A41SK0F-Arduino.

El pin 1 del sensor conectado al pin analógico A0 de la tarjeta Arduino, el pin 2 a tierra (GND) y el pin 3 a 5V.

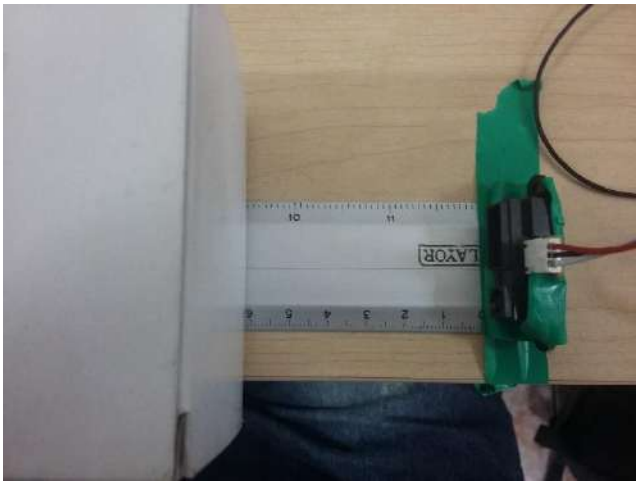

Figura 21. Prueba sensor de distancia.

Se coloca un objeto aproximadamente a 6cm de distancia del sensor como se muestra en la Figura 21, la tarjeta Arduino lee el voltaje analógico que proporciona el sensor y utiliza la ecuación cúbica de interpolación para calcular la distancia a la que se encuentra el objeto, en la Figura 22 se muestran las lecturas del sensor a través del monitor serie de Arduino en intervalos de 0.5 segundos.

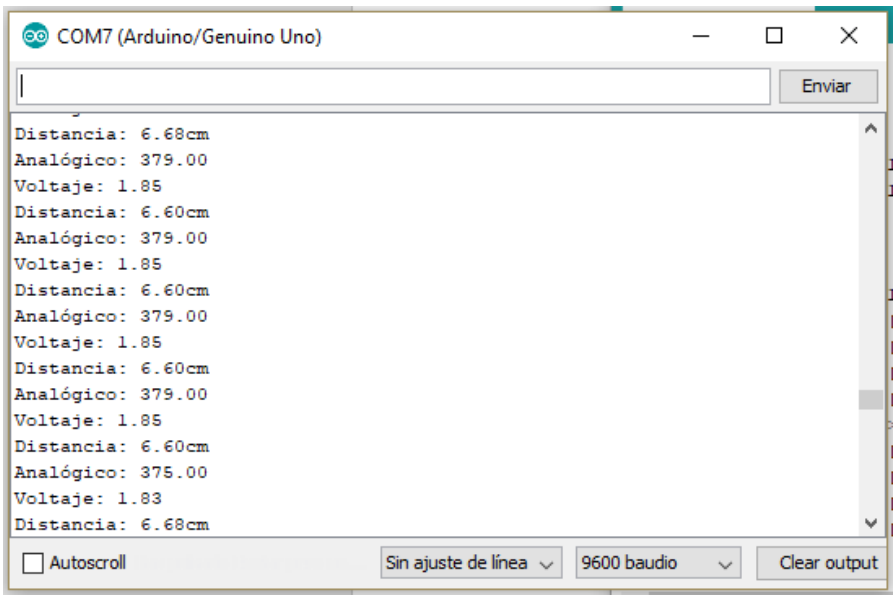

Figura 22. Respuesta del Sensor Sharp GP2Y0A41SK0F en el monitor serie del IDE Arduino.

Las lecturas que se observan en la Figura 22 no son exactas, pues la interpolación cúbica da como resultado una aproximación al valor real de la distancia a la que se encuentra el objeto, sin embargo, el resultado obtenido por medio de la ecuación cúbica es muy bueno.

3.4.3 Control de Focos de AC con relevadores

La tercera interfaz que se presenta sirve para el control de dispositivos de alto consumo de energía (potencia), por medio de relevadores. En esta sección se detalla cómo controlar un foco de AC por medio de un relevador y una tarjeta Arduino. En la Figura 23 se muestra el esquema de conexión del foco al relevador, y del relevador a la tarjeta Arduino.

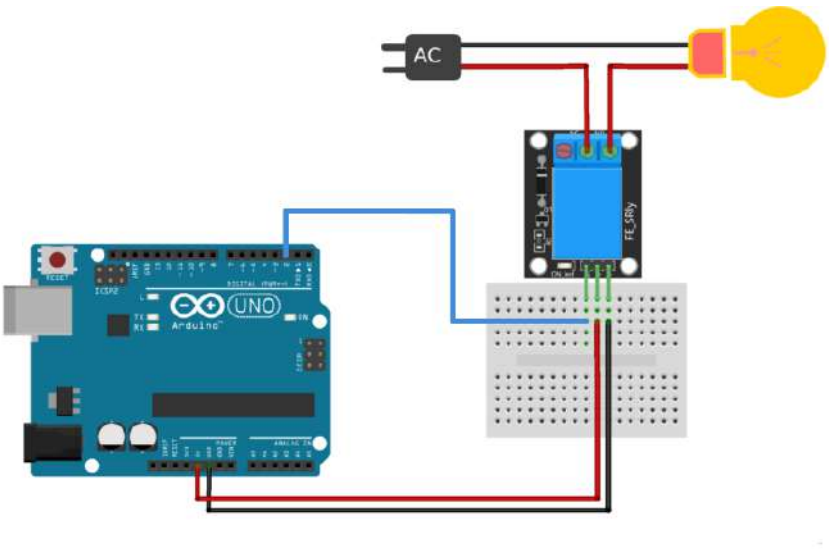

Figura 23. Esquema de control de un foco de AC con relevador y tarjeta Arduino.

El relevador cuenta con 3 pines, un pin para alimentación (5V), un pin para tierra (GND), un pin de señal (S) y cuenta con tres conectores, un conector denominado normalmente abierto (NA), un conector denominado normalmente cerrado (NC), y un conector denominado común (C).

Al enviar una señal al pin S el relevador se comportará como un interruptor, pues en caso de que la señal sea de 5V, por ejemplo, permitirá el flujo de energía entre el conector C y el conector NC, si se cambiará el valor de la señal a 0V, el relevador permitiría el flujo de energía entre el conector C y el conector NA. Es por esta razón que en el esquema presentado en la Figura 23, se muestra uno de los cables del foco cortado a la mitad, y conectando una punta al conector C y otra punta al conector NA, para que al alimentar al relevador el foco no se encienda al instante, sino hasta que al relevador se le envíe una señal de 0V. Las pruebas para esta interfaz se mostrarán en el apartado 3.6 de pruebas de integración.

3.5 Interfaz de comunicación Raspberry – Microcontrolador

Es necesario y buena práctica delegar funcionalidades y tareas a diferentes dispositivos, pues así el sistema continúa siendo escalable, mantenible, modular y descentralizado. Se opta por delegar las tareas de monitoreo y control de los sensores y actuadores, a las tarjetas Arduino para que la tarjeta Raspberry se encargue del procesamiento de voz, síntesis de voz y toma de decisiones y que de esta manera el desgaste a nivel de

hardware sea menor, algo que será muy benéfico al performance (desempeño) del sistema. Para realizar esto se hace uso de comunicación serie entre la tarjeta Raspberry y la tarjeta Arduino.

3.5.1 Comunicación serie

La comunicación serie es un proceso en el que se transmiten datos bit a bit, de forma secuencial por medio de un canal de comunicación o bus de datos. Este proceso puede dividirse en dos categorías: **síncrona** o **asíncrona**.

La transmisión síncrona requiere de una señal de reloj y una línea de datos, mientras que en la transmisión síncrona no se envía la señal de reloj, por lo que el emisor y el receptor deben operar a la misma frecuencia y fase. Se utiliza transmisión síncrona cuando la distancia entre receptor y emisor es pequeña, mientras que para distancia mayores se utiliza transmisión asíncrona (García Breijo, 2008).

3.5.2 Protocolo de comunicación serie UART

El protocolo Transmisor-Receptor Asíncrono Universal (UART) se caracteriza por tener mantener dos líneas de comunicación, una para trasmitir datos y otra para recibir datos, lo que permite implementar este protocolo en modo **full-duplex** (envío y recepción simultanea).

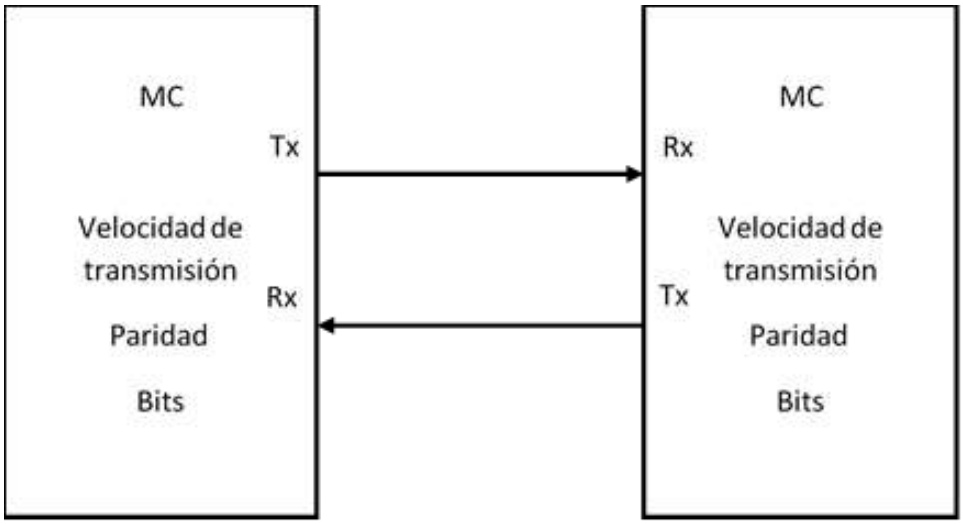

Figura 24. Esquema de conexión para el protocolo UART.

Para configurar el protocolo de comunicación serie entre dos dispositivos, es necesario que ambos tengan la misma configuración, principalmente en parámetros como la velocidad de transmisión (baudrate), paridad (verifica si hay errores en la transmisión serie), bits de datos (cantidad de bits de transmisión). La línea de transmisión (Tx) debe estar conectada a la línea de recepción (Rx) esto se puede ver con más claridad en la Figura 24.

3.5.3 Protocolo de comunicación I2C

El protocolo Interfaz-Inter Circuitos (I2C), es un bus que se basa en la comunicación a través de 2 hilos, uno para la señal de reloj SCL y otro para transmitir los datos SDA, por lo que el modo de operación es **half-duplex** (envío o recepción) y además es síncrono (García Breijo, 2008). En el bus puede haber dispositivos operando en tres modos diferentes:

1. Maestro: Dispositivo que tiene la iniciativa en la transferencia de datos, decide con que dispositivo realiza la transferencia, el sentido de la transferencia, y cuando finaliza. Es quien genera la señal de reloj.
2. Esclavo: Dispositivo conectado al bus I2C, esperando que algún dispositivo maestro intente comunicarse con este.
3. Multimaestro: Hay más de un dispositivo maestro en el bus, cada dispositivo maestro puede funcionar como esclavo también.

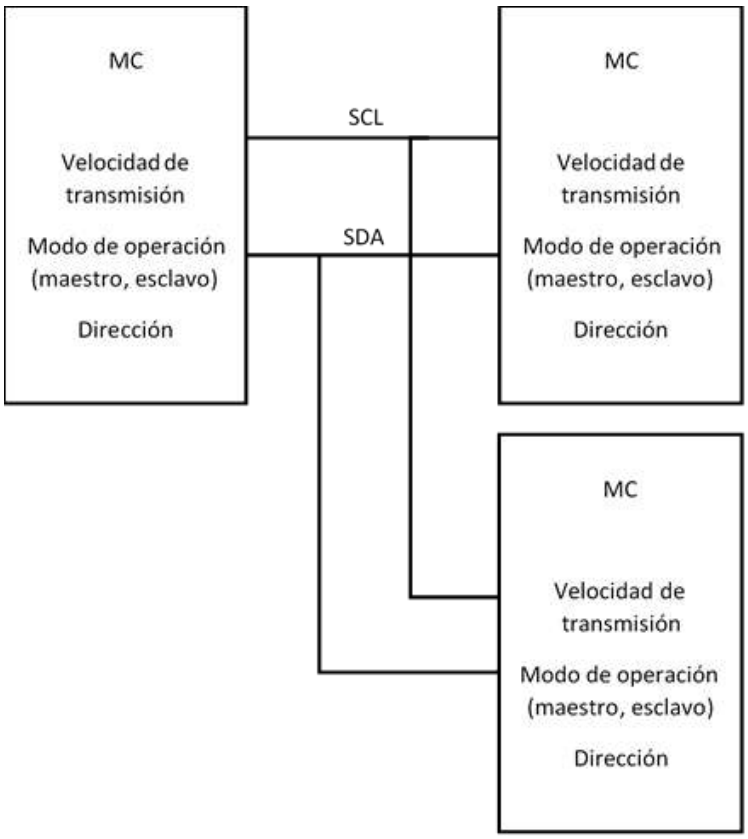

Figura 25. Esquema de conexión para el protocolo I2C.

Los dispositivos conectados al bus I2C, deben transmitir a la misma velocidad, debe haber al menos un dispositivo configurado como maestro, y cada dispositivo debe tener asignada una dirección. A diferencia del protocolo UART, donde las líneas de transmisión y recepción se conectaban de forma cruzada, en I2C las líneas SCL (señal de reloj) de los dispositivos se conectan entre sí, y de igual manera las líneas SDA (línea de datos). El esquema básico de conexión para este protocolo se muestra en la Figura 25.

3.5.4 Implementación del protocolo UART entre Arduino y Rapsberry

Se opta por implementar el protocolo UART debido a que se requiere una comunicación tipo **full-duplex,** pues el asistente debe ser capaz de enviar instrucciones a la tarjeta Arduino y en el caso de que algún sensor este proporcionando lecturas anormales, la tarjeta Arduino pueda enviar alertas a la tarjeta Raspberry en cualquier momento.

En Python existe la librería “serial”, la cual provee la clase y métodos necesarios para implementar el protocolo UART, es necesario importar la

librería y obtener una instancia de la clase Serial utilizando las siguientes instrucciones:

```
import serial
ser = serial.Serial('/dev/ttyACM0', 9600, timeout=8)
```

El primer parámetro es el nombre del puerto serie que se quiere abrir, el segundo parámetro es la velocidad de transmisión (bits/segundo) y el tercer parámetro es el tiempo de espera de lectura (en segundos). De esta manera se tiene ya una instancia configurada, sólo resta abrir el puerto serie y para esto, primero se valida si el puerto ya se encuentra abierto. Si el puerto ya está abierto se cierra y se vuelve a abrir, de lo contrario sólo se procede a abrirlo. Se hace uso de las siguientes instrucciones:

```
if (ser.isOpen):
  ser.close()
  ser.open()
else:
  ser.open()
```

Es necesario ejecutar estas instrucciones repetidamente hasta que se detecte un dispositivo conectado al puerto serie que se está intentando abrir, de no hacerse así, si no se detecta algún dispositivo la primera vez que se ejecutan las instrucciones, el asistente virtual iniciaría sin tener abierto un puerto serie y por ende no habría forma de comunicarse a la tarjeta Arduino para que este ejecute las tareas de automatización.

Para enviar instrucciones a la tarjeta Arduino, se hace uso de la siguiente instrucción:

```
ser.write('.....\n')
```

Previo al envío de la instrucción, el asistente virtual debió haber validado alguna instrucción dictada por el usuario. Se envía una cadena de texto, puede enviarse un código o un número e incluir el carácter '\n' para indicar el fin de la línea (EOL). Para estos casos se enviará un número, el cual será validado por la tarjeta Arduino para ejecutar una tarea asociada a dicho número. Este tipo de operaciones, se abarcan en el segundo flujo de operación del asistente virtual, que es el flujo de automatización el cual se muestra a detalle en la Figura 26.

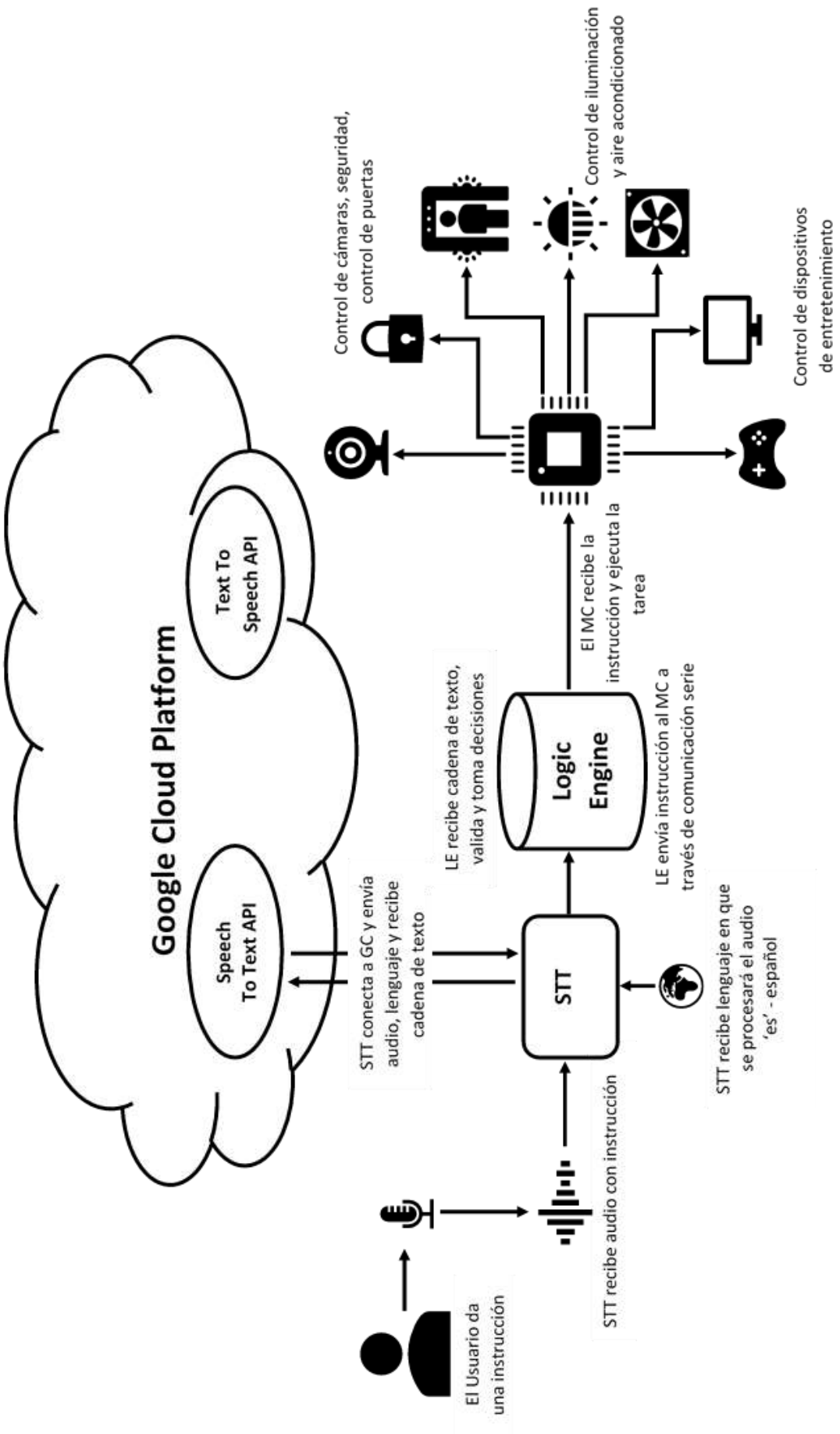

Figura 26. Flujo de automatización del asistente virtual.

Un usuario indica al asistente virtual activar algún dispositivo o aparato del hogar, el asistente valida qué se debe activar y envía un código a la tarjeta Arduino, código relacionado a la tarea que se debe ejecutar, la tarjeta Arduino recibe el código, lo valida y ejecuta la tarea.

La configuración del protocolo en la tarjeta Arduino es también muy simple, para inicializar el puerto serie en Arduino se hace uso de la siguiente instrucción:

```
Serial.begin(9600);
```

Sólo es necesario parametrizar con la velocidad de transmisión. Para la recepción de datos, se puede hacer uso de una rutina SerialEvent, la cual se ejecuta entre cada ciclo de ejecución del programa principal. Un evento en el puerto serie ocurre cuando un nuevo dato llega a la terminal RX del puerto serie de la tarjeta Arduino. Mientras el puerto serie esté disponible, se intentará leer datos de este puerto byte a byte, en el momento en que se lee un carácter '\n' (EOL) se cambia el estatus de una variable bandera, indicando que se ha terminado de recibir un paquete de datos un ejemplo del código se presenta en la Figura 27.

```
void serialEvent(){
  while (Serial.available()){
    char inChar = (char)Serial.read();
    inputString += inChar:

    if(inChar == '\n'){
      stringComplete = true;
    }
  }

}
```

Figura 27. Implementación de la rutina SerialEvent.

La rutina SerialEvent no está disponible para todas las tarjetas Arduino, pero sí para el Arduino Uno. La documentación completa de esta rutina se encuentra en el sitio oficial de Arduino.

Llegado este punto se tiene comunicación de la tarjeta Raspberry hacia la tarjeta Arduino, por lo que resta implementar la lectura de datos del

puerto serie en la tarjeta Raspberry e implementar la escritura de datos en puerto serie en la tarjeta Arduino.

Para terminar de implementar el protocolo de comunicación serie entre las Tarjetas Raspberry y Arduino y que se tenga una comunicación bidireccional full-duplex, en la Tarjeta Arduino se hace uso de las siguientes instrucciones:

```
Serial.print(“”); //Cadena de texto sin salto de línea

Serial.println(“”); //Cadena de texto con salto de línea
```

En esta implementación se hace uso de la segunda instrucción, pues para el asistente virtual en la tarjeta Raspberry, el carácter de salto de línea es un indicador de finalización de trama, necesario para leer del puerto serie.

La tarjeta Arduino estará monitoreando sensores, y validando que los valores que se obtienen de estos se encuentren en un rango específico, si el valor de algún sensor saliera del rango se enviará una trama (alerta) al asistente virtual indicando la variable física de la cual se están obteniendo valores anormales y el valor obtenido. La trama emitida por la tarjeta Arduino tiene el siguiente formato sin los caracteres **“|”**:

“*magnitud_fisica*|*valor_obtenido*|*\n*”

Dónde:

- magnitud_fisica: Es un carácter que indica la variable física que se está midiendo, por ejemplo, **t** para temperatura, **h** para humedad o **d** para distancia.
- valor_obtenido: Es el valor obtenido del sensor.
- \n: Es el carácter de salto de línea (EOL) o indicador de finalización de trama.

Dado que la tarjeta Raspberry no tiene la capacidad de utilizar interrupciones para la recepción de información, se hace uso de un **Thread**, que permite ejecutar instrucciones de manera simultánea al programa principal, para monitorear constantemente el estado del puerto serie a la espera de datos que puedan ser leídos. Se hace uso del módulo threading,

el cual permite la implementación de Threads (Hilos), se importa de la siguiente manera:

```
import threading
```

Se implementa una clase “SerialPortListenerThread” con su método el principal **run** en el cual se implementa un ciclo infinito de escucha para el puerto serie.

Cuando el asistente virtual recibe una trama en el puerto serie, verifica que la trama no esté vacía, esto para invocar al método encargado de validar las tramas recibidas por puerto serie para reportarlo al usuario.

```
while True:
        input_stream = ser.readline()
        if input_stream != ‘’:
                check_data_port(input_stream)
                ser.flush()
```

La implementación de la clase “SerialPortListenerThread” para monitoreo de puerto serie se realiza en el componente main.py, pues el proceso secundario que monitorea el puerto serie debe ejecutarse desde el momento en que inicia el sistema, el Hilo (Thread) se instancia y se ejecuta con las siguientes instrucciones:

```
thread_listen_port = SerialPortListenerThread(1, "Thread-Port Listener", ser)

thread_listen_port.start()
```

La clase “SerialPortListenerThread” se instancia con un id, un nombre y una instancia del puerto serie configurada e iniciada. El Thread se ejecuta al invocar el método start(), el cual automáticamente ejecuta el método **run** encargado de monitorear el puerto serie.

La recepción de tramas por parte del asistente virtual permite que se cumpla el tercer flujo de operación, el flujo de monitoreo. Este flujo fue diseñado para que el asistente virtual pueda reportar eventos o lecturas inusuales detectadas por medio de los sensores al usuario, se puede observar a detalle en la Figura 28.

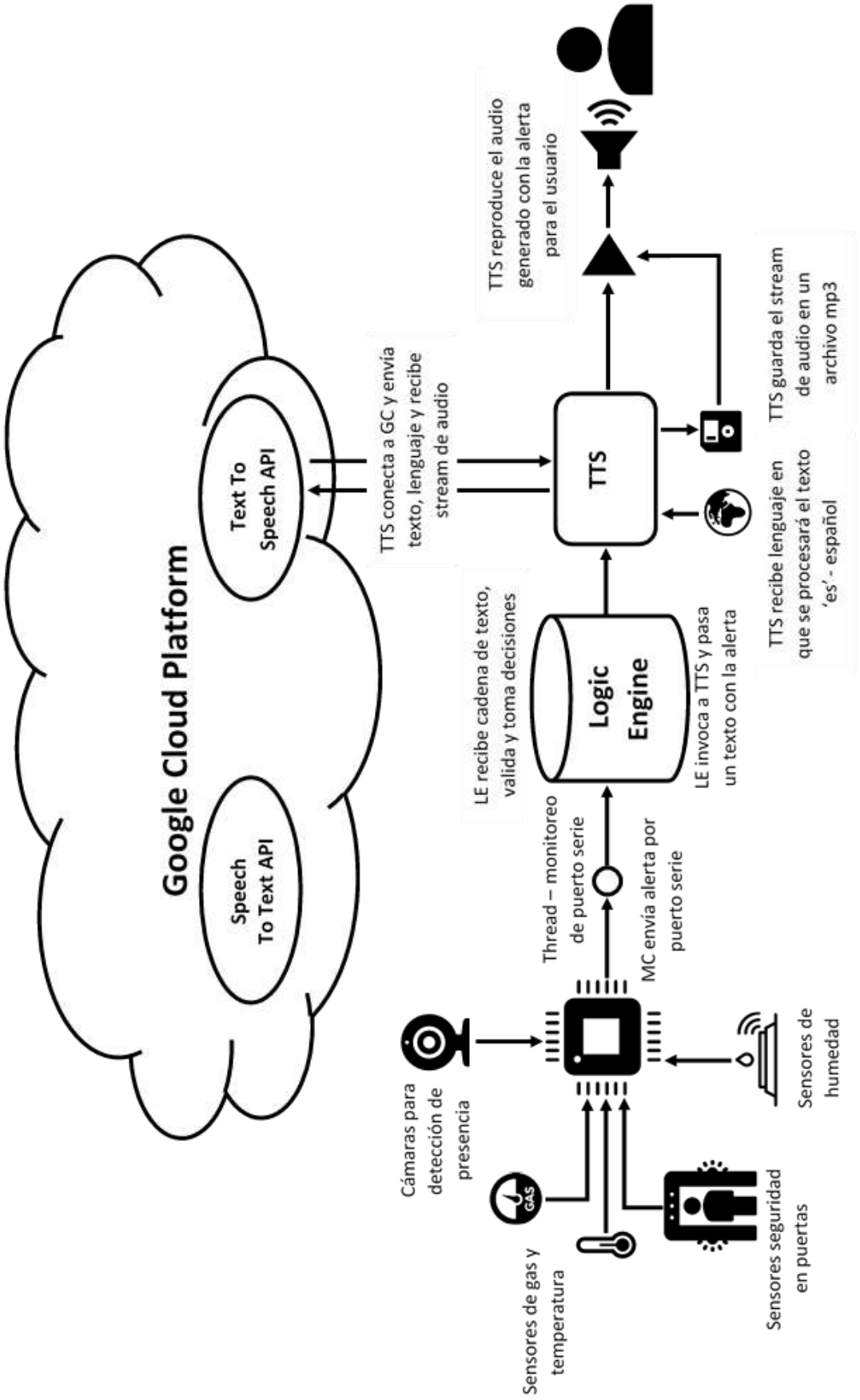

Figura 28. Flujo de monitoreo del asistente virtual.

3.6 Pruebas de integración

Una vez que se ha integrado al software los distintos flujos de operación, el asistente virtual valida las instrucciones indicadas por el usuario y/o alertas procedentes de la tarjeta Arduino a través de comunicación serial. Todas las instrucciones sin importar su procedencia, se validarán en el Motor Lógico, en el cual se implementa la toma de decisiones para procesar una instrucción de acuerdo al flujo de operación al que pertenece.

La primera prueba que se realizó fue con un sensor DHT03 (sensor digital, Figura 29) de temperatura y humedad para el flujo de monitoreo. La prueba consistió en lo siguiente: En cuanto la tarjeta Arduino obtuviera una lectura de temperatura o humedad en un rango especificado, se envía al asistente virtual por puerto serie una trama de datos indicando si se trataba de humedad o temperatura, y el valor que se estaba obteniendo del sensor. La trama proveniente de la tarjeta Arduino es recibida por el asistente virtual y este sintetiza un audio con indicando al usuario “Se ha detectado una temperatura de x grados centígrados”, y en el caso de humedad “Se ha detectado una humedad de x %”, donde x es el valor correspondiente.

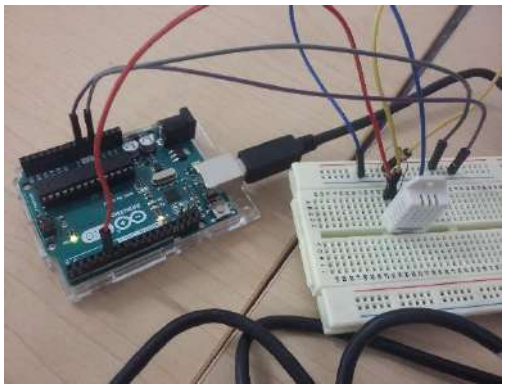

Figura 29. Sensor DHT03 conectado a la tarjeta Arduino.

El asistente virtual recibe la trama con el formato **“xn\n”**, donde **x** indica la variable física que se está monitoreando, **n** es el valor de la variable física y **\n** es el carácter de fin de línea (EOL) necesario para que la trama pueda ser leída del puerto serie. En la Figura 30, se muestra la trama enviada por la tarjeta Arduino y la cadena de texto que será sintetizada a voz.

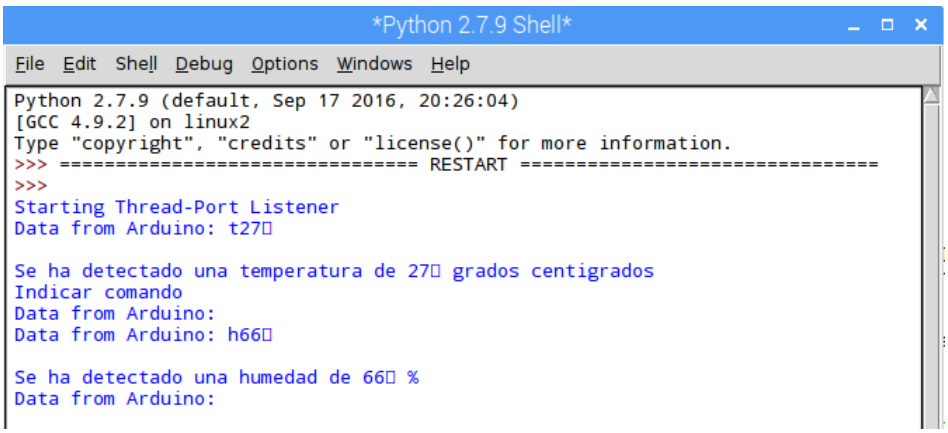

Figura 30. El asistente virtual recibe tramas con valores de temperatura y humedad.

La segunda prueba se realizó con un sensor analógico para medir distancia Sharp GP2Y0A41SK0F, la prueba consistió en lo siguiente: Se valida la aproximación de algún objeto hacia este sensor como se muestra en la Figura 31, en el momento en que la tarjeta Arduino obtiene una lectura que indica que el objeto se encuentra a una cierta distancia (en este caso 6 cm), se envía al asistente virtual una trama de datos indicando la distancia a la que se detectó el objeto. De igual manera que en la prueba uno, el asistente virtual recibe una trama de datos y sintetiza un audio indicando al usuario “Se detectó un objeto a x cm de distancia" del sensor (colocado en una zona de interés).

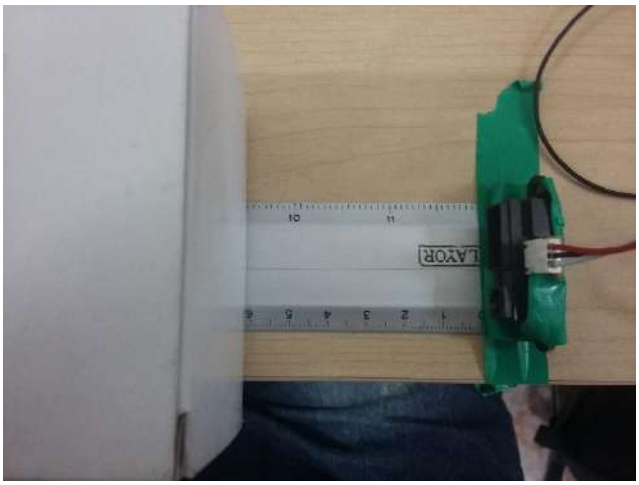

Figura 31. Objeto colocado aproximadamente a 6 cm de distancia del sensor.

Al igual que en la primera prueba, se recibe una trama con el formato **“xn\n”**, Figura 32.

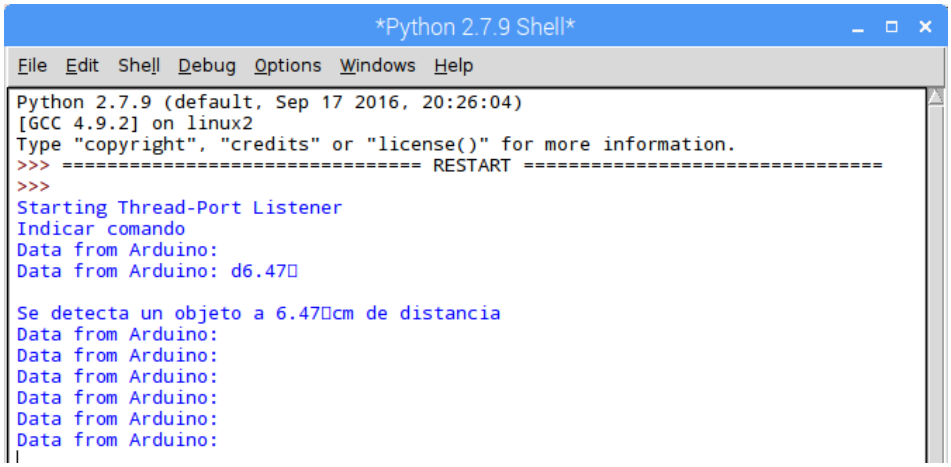

Figura 32. El asistente virtual recibe trama con valor de distancia.

Para la tercera prueba, se hace uso del módulo de relevadores, esto para encender un foco conectado a una toma de corriente alterna. Un usuario dicta la instrucción “luces” y el asistente virtual valida y envía a la tarjeta Arduino un código para indicarle que active el relevador que permitirá que el foco sea encendido. Para apagar el foco se dicta la misma instrucción.

En la Figura 33 se muestran las instrucciones reconocidas por el asistente virtual para encender y apagar el foco.

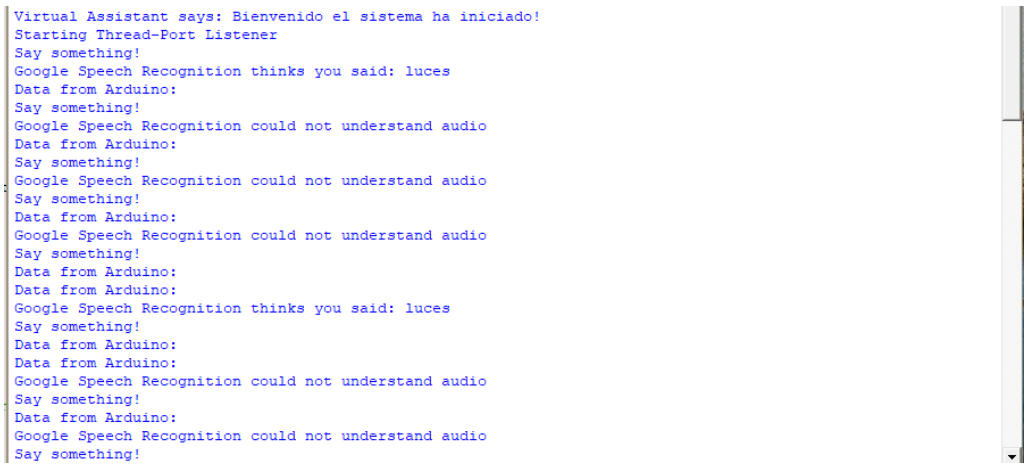

Figura 33. Instrucción “luces” para encender y apagar foco.

En la Figura 34 a) se muestra el foco encendido después de validar por primera vez la instrucción “luces”. En la Figura 34 b) se muestra el foco apagado después de validar por segunda vez la misma instrucción.

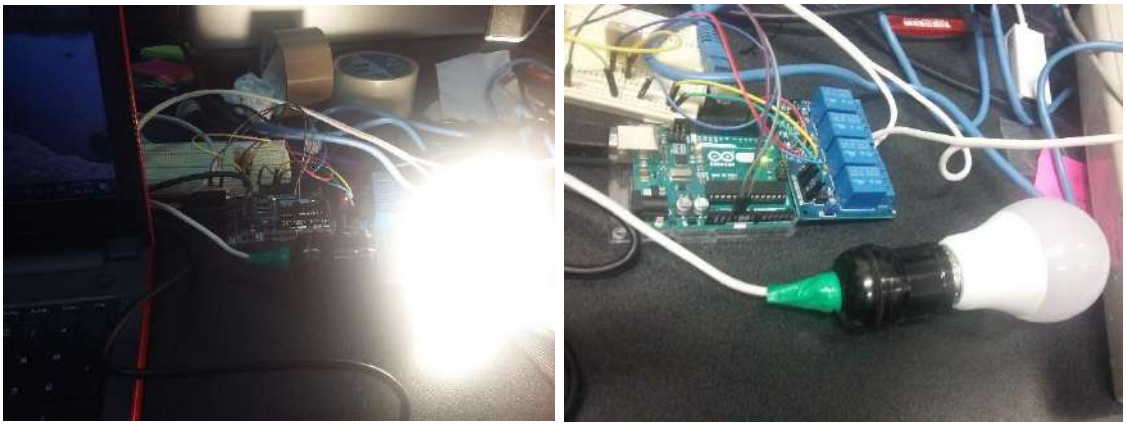

Figura 34. a) Foco encendido al validar por primera vez, b) Foco apagado al validar por segunda vez.

Como cuarta prueba se hace la consulta del concepto de electrónica para que el asistente proporcione una respuesta, indicando el siguiente comando: “define electrónica”. En la Figura 36 se muestra la instrucción recibida por el asistente virtual y la respuesta que dictará al usuario.

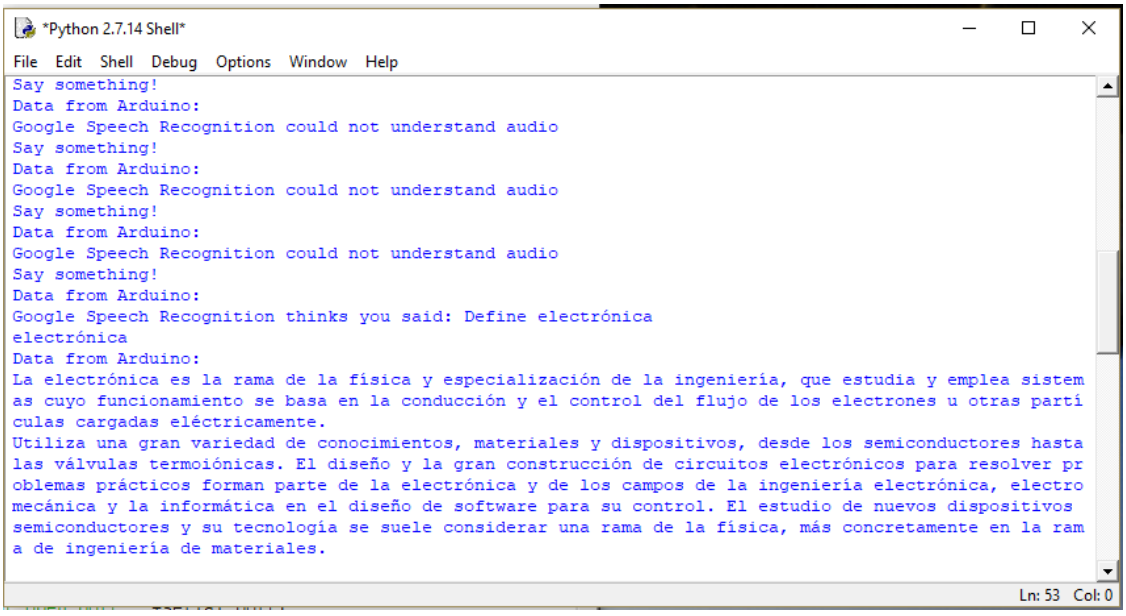

Figura 36. El asistente virtual recibe la instrucción “define electrónica”.

La tercera y cuarta prueba se realizaron en el sistema operativo Windows, por la limitación del poco alcance del micrófono adquirido para la tarjeta Raspberry.

Las pruebas uno y dos comprenden el flujo de operación de monitoreo, la prueba tres pertenece al flujo de automatización y la prueba cuatro al flujo de un asistente convencional.

3.7 Primera fase de pruebas: Análisis de resultados, correcciones y mejoras

Durante las pruebas, se detectó un problema en la comunicación serie entre el asistente virtual y la tarjeta Arduino, pues si por alguna razón la tarjeta Arduino se desconectaba, el asistente virtual lanzaba una excepción y se interrumpía la ejecución. Este problema se resolvió utilizando una estructura try-except, atrapando la excepción que ocurría al intentar abrir el puerto serie. Se utilizó la siguiente estructura:

```
try:
    ser = serial.Serial(name_port, 9600, timeout=5)
except IOError :
```

El código anterior fue encapsulado en un bucle para que el asistente virtual siguiera ejecutándose hasta que la tarjeta Arduino fuera conectada y el puerto serie pudiera ser abierto.

Se detectó también una excepción que se generaba cuando el asistente virtual no podría leer datos del puerto serie, para lo cual se utilizó otro bloque try-except para atrapar esta excepción. Se utilizó la siguiente estructura:

```
try:
    input_stream = self.port.readline()
except serial.SerialException:
```

Durante las pruebas con todos los flujos de operación integrados, ocurría una colisión de flujos, esto entre el flujo de operación convencional y el flujo de monitoreo, esto cuando ambos flujos llegaban al componente TTS, pues habiendo una síntesis de voz en progreso, no se podía iniciar otra lo que ocurría que la ejecución del asistente virtual se interrumpiera.

Para resolver este problema al finalizar cada síntesis de voz, se cierra la instancia de la API, de esta manera se manejaban instancias diferentes, contrario a lo que se hacía al principio que era abrir una sola instancia para ambos flujos. Además, se utilizó también un bloque try-except para atrapar las excepciones ocurridas por ejecución utilizando la instrucción except RuntimeError.

Los anteriores detalles fueron solucionados y las pruebas de integración, son las que se han presentado en la sección 3.6. Estas correcciones mejoraron la ejecución del asistente virtual, pues ahora su ejecución no si interrumpe salvo por conexión a internet que es la principal dependencia.

4. RESULTADOS

Durante el desarrollo del prototipo y la fase de pruebas se detectaron varias limitaciones las cuales no son críticas, pero de ser solventadas, hace que el asistente virtual pueda tener un mejor performance, lo suficiente como para tratarlo como una versión beta.

Esta versión del prototipo es usable, aunque para tener un buen rendimiento es indispensable una conexión a internet veloz, y un micrófono de gran alcance.

Quedó demostrado que es factible la aplicación de asistentes virtuales en áreas como la domótica, pues hay tareas que se pueden automatizar muy bien.

Acercamiento a la comunidad de desarrolladores de software e investigación a la implementación de asistentes virtuales para aplicarlos en el entorno en el que nos desenvolvemos.

5. CONCLUSIONES Y TRABAJO A FUTURO

Estamos en una época en la que hay gran diversidad de tecnologías, que por sí solas son sorprendentes, novedosas y muy interesantes, pero ¿Qué pasa cuando convergen dos o más tecnologías?, el resultado es sinergia tecnológica. Esta sinergia de la que se habla está al alcance de cualquier persona con interés y ganas por crear e innovar.

En nuestros días existen muchas tecnologías *OpenSource*, las cuales podemos tomar, modificar y/o adaptar a nuestras necesidades para crear nuevas cosas. Estas tecnologías aguardan a ser explotadas para mejorar nuestro entorno. La propuesta que se presenta aquí, en un futuro será algo muy común.

Se pretende a futuro, implementar un procesador de lenguaje natural y un sintetizador de texto a voz, para eliminar la dependencia de internet y aumentar los tiempos de respuesta del asistente virtual. Se tiene planeado incorporarle al asistente virtual la capacidad de aprendizaje, esto utilizando técnicas de inteligencia artificial. Se busca además sustituir el protocolo de comunicación serie por un protocolo TCP/IP para montar al asistente virtual en una red.

Los asistentes virtuales pueden ser implementados en cualquier lugar, en casas, oficinas, hospitales, departamentos de policías, escuelas, etc. Entre las aplicaciones de gran aporte que se prevén están: como medio de monitoreo y cuidado para pacientes y/o personas de la tercera edad, como medio de ayuda en vías públicas como paradas de autobuses y en escuelas como una herramienta de enseñanza.

6. REFERENCIAS

Aceves, M. A., & Ramos, J. M. (2012). *Fundamentos de Sistemas Embebidos.* México: Asociación Mexicana de Mecatrónica A.C.

Adafruit (2015). Obtenido de https://github.com/adafruit/Adafruit_Sensor

Amazon. (14 de Diciembre de 2016). *Alexa skills kit*. Obtenido de Amazon developer: https://developer.amazon.com/alexa

Apple. (2017). *Siri*. Obtenido de https://www.apple.com/ios/siri/

Arduino. (2017). Obtenido de https://www.arduino.cc/en/Guide/Introduction

Arduino Uno. (2017). Obtenido de https://store.arduino.cc/usa/arduino-uno-rev3

Arduino Leonardo. (2017). Obtenido de https://store.arduino.cc/usa/arduino-leonardo-with-headers

Corona Ramírez, L., Abarca Jiménez, G., & Mares Carreño, J. (2014). *Sensores y actuadores*. Distrito Federal: Larousse - Grupo Editorial Patria.

Cox, T. (2014). *Raspberry Pi cookbook for Python programmers.* Birmingham, UK: Packt Pub.

Díaz Guerrero, R. E. (2016). *Digitalización de sólidos utilizando análisis de franjas y técnicas de desplazamiento de fase en arquitecturas ARM (Maestría). Universidad Autónoma de Querétaro, Facultad de Informática.* Queretaro, México.

Electrónica Embajadores. (2017). Obtenido de https://www.electronicaembajadores.com/datos/pdf1/ss/sshu/RHT03.pdf

García Breijo, E. (2008). *Compilador C CCS y simulador PROTEUS para microcontroladores PIC.* México: Alfaomega.

Google. (2017). *Google Now.* Obtenido de http://www.google.com/landing/now/

González M, I. (2006). *Coprocesadores Dinámicamente Reconfígurables en Sistemas Embebidos basados en FPGAs (Doctorado).* Universidad Autónoma de Madrid, Departamento de Ingeniería Informática.

Hauswald, J., Laurenzano, M. A., Zhang, Y., Li, C., Rovinski, A., Khurana, A., ... & Mars, J. (2015, March). Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers. In *ACM SIGPLAN Notices* (Vol. 50, No. 4, pp. 223-238). ACM.

Hoile, C., Bowman, C., Meijer, S. D., Corteil, B., & Orsini, L. (2014). *Make: Raspberry Pi and AVR Projects.* Sebastopol, California: Maker Media.

Huidobro, J. M., & Millán, R. J. (2010). *Manual de Domótica.* España: Creaciones CopyRight.

Inteligencia Artificial. (2017). Obtenido de http://www.inteligenciaartificialyrobotica.com/esp/item/495/11/relay-module4

Mechling, L. C., Gast, D. L., & Seid, N. H. (2009). Using a personal digital assistant to increase independent task completion by students with autism spectrum disorder. *Journal of autism and developmental disorders*, *39*(10), 1420-1434.

Microsoft. (2017). *Cortana*. Obtenido de https://www.microsoft.com/es-es/windows/Cortana

Mishra, A., Makula, P., Kumar, A., Karan, K., & Mittal, V. K. (2015, May). A voice-controlled personal assistant robot. In *Industrial Instrumentation and Control (ICIC), 2015 International Conference on* (pp. 523-528). IEEE.

Naylamp Mechatronics SAC. (2016), Obtenido de http://www.naylampmechatronics.com/modules//smartblog/images/40-single-default.jpg

Pant, T. (2016). *Building a Virtual Assistant for Raspberry Pi.* Ghaziabad, Uttar Pradesh, India: Apress.

Pérez A., D. A. (Octubre de 2009). Obtenido de Academia: https://www.academia.edu/16523506/Info_Sistemas_Embebidos

Pololu Robotics & Electronics. (2018). Obetenido de https://www.pololu.com/file/0J713/GP2Y0A41SK0F.pdf

Poole, M. (2015). *Building a Home Security System with Raspberry Pi Build.* Birmingham, UK: Packt Publishing Ldt-.

Riley, M. (2012). *Programming Your Home.* Raleigh, USA: The Pragmatic Bookshelf.

Romero Morales, C., Vázquez Serrano, F., & Castro Lozano, C. (2011). *Domótica e inmótica*. México: Alfaomega.

UpSkill Learning (2016). *Raspberry Pi 3 And BeagleBone Black for Engineers*. CreateSpace Independent Publishing Platform.

Upton, E. (2016). *Raspberry Pi user guide.* Chichester, United Kingdom: John Wiley & Sons.

Warner, T. (2013). *Hacking Raspberry Pi.* Indianapolis, IN: Que Publishing.

Watkiss, S. (2016). *Learn electronics with Raspberry Pi.* Redditch, UK: Apress.

WolframAlpha. (2018). Obtenido de https://www.wolframalpha.com/