Universidad ORT Uruguay
Facultad de Ingeniería

Documentación final

Guardian Engine

Ariel Sadi - 203590

Juan Bautista Martinez - 235608

Tutores

Emiliano Espindola

Juan Pablo Silva

ÍNDICE

1. Introducción

El presente documento tiene como fin servir de registro y documentación del desarrollo de un proyecto que gira en torno al concepto de internet de las cosas (IoT). Este informe pretende dar información detallada del funcionamiento de cada componente que esté incluido en el sistema además de relatar los acontecimientos y percances que llevaron a tomar ciertas decisiones sobre el prototipo final.

El proyecto en cuestión lo denominamos “Guardian Engine”, y consiste en un sistema de seguridad integrado para un vehículo automotor. La idea es que la conexión entre sus elementos, y estos con el usuario sea por internet. Para establecer y dirigir esta conexión usamos un programa libre de uso llamado “Thingsboard”, el cual es un software que funciona como nube y permite al cliente estar al tanto de la seguridad de su auto, incluso estando lejos de él.

1.2 Problema

En Uruguay, 42 vehículos son robados al día (según radio sarandí en enero de 2021), de los cuales menos del 3% de ellos terminan con alguna persona detenida por el delito. Es decir, en otras palabras, si te roban el auto hay menos de 3% de chances de que detengan al culpable y te devuelvan el vehículo y todas tus posesiones.

Frente a este problema nosotros vimos una oportunidad, de ahí surge la idea de crear un sistema de seguridad que pueda prevenir este tipo de delitos.

Nosotros planteamos 3 problemas en concreto a resolver para el cliente:

1. Tener evidencia fotográfica del delincuente en caso de que haya robo del auto, para no ser parte del 97% que no recibe justicia por el delito.
2. Tener un mecanismo que impida que el delincuente se pueda ir con el auto.
3. Informar inmediatamente que se reporte alguna violacion de seguridad del vehículo.
4. Poder controlar el auto a grandes distancias, de tal forma que no tenga que ir al auto en caso de que haya un reporte.

1.3 Solución

Guardian Engine plantea brindar un sistema de seguridad adicional al auto. Pretende dar tranquilidad al chofer de no ser víctima de un delito.

Se trata de un sistema de seguridad integrado conectado al internet, que permite al usuario controlar la seguridad de su auto desde cualquier lugar del mundo. Mediante componentes electrónicos, la idea principal de Guardian Engine es proveer de mecanismos que prevengan el robo, en el mejor de los casos, y que además registre evidencia fotográfica en el caso de que el auto haya sido robado, en el peor de los casos.

En concreto, el sistema plantea brindar soluciones a los 4 problemas de la siguiente manera:

1. Para la evidencia fotográfica utilizaremos una cámara que registre con fotos el interior del auto en caso de percibir movimiento.
2. Para impedir que el delincuente prenda el auto, utilizaremos una

válvula que impida el paso de combustible al motor en caso de un reporte de movimiento.
3. La conexión a internet permitirá informar al cliente, mediante una aplicación, de la situación actual del vehículo.
4. La misma conexión también permitirá al usuario controlar el auto en caso de ser precisado. Como es por internet, la conexión es remota y accesible en todo el mundo, siempre que se tenga internet.

2. Objetivos

En el anteproyecto de Guardian Engine, pudimos establecer los objetivos a cumplir en el desarrollo del proyecto. Está sección pretende recordarlos para poder analizar su cumplimiento o no en la conclusión de este documento.

Los mismos son:

- Crear un sistema inteligente de seguridad IoT completamente funcional para un vehículo, conectado a la nube para cumplir su función. El mismo deberá seguir las características detalladas de la sección descripción.

- Debe ser totalmente funcional en el mercado de seguridad. Práctico y realista, pero a la vez lo suficientemente innovador como para captar la atención de posibles clientes e inversores.

3. Descripción y detalles del proyecto

Esta sección se encargará de plantear las distintas partes que conforman al prototipo y sus funcionalidades.

3.1 Descripción de la solución

El sistema es una integración para un vehículo con el motivo de protegerlo contra hurtos. Cuenta con varios dispositivos que permiten realizar las siguientes acciones:

- Detectar movimiento para accionar otros dispositivos
- Tomar fotos del interior del auto
- Procesar las fotos para reconocer individuos dentro del mismo
- Bloquear las puertas
- Bloquear el flujo de combustible
- Procesar e informar la ubicación del vehículo
- Impedir la visión dentro del vehículo con humo

3.2 Componentes

Todos los componentes mencionados integran una parte de la solución de nuestro sistema, cabe mencionar que luego se profundizará más en cómo se llevaron a cabo y en otros componentes que se agregaron para su funcionalidad.

Todos ellos, excluyendo la cámara, son controlados utilizando un microprocesador Node ESP8266, que nos proporciona la utilidad para controlar los componentes con una superficie bastante pequeña, además de contar con un módulo WiFi y ser compatible con el IDE de Arduino.

3.2.1 Sensor de movimiento

El sensor de movimiento es un componente que consideramos fundamental para nuestro proyecto, debido a que es el sensor es el primer determinante de actividad sospechosa dentro del vehículo. Utilizamos un sensor PIR HC-SR501, que es diminutivo para Passive Infrared. El mismo reacciona ante ciertos niveles de radiación, mientras mayor temperatura tenga mayor radiación emitirá. Está dividido en dos zonas con el motivo de obtener la diferencia en el movimiento, por lo que es ideal para detectar individuos ingresando a una zona.

3.2.2 Válvula

Utilizar una válvula controlable fue uno de los puntos que consideramos agrega bastante seguridad a la hora de proteger el vehículo ante hurto. Es por esto que elegimos utilizar una válvula solenoidal controlable mediante tensión.
Funciona como interruptor para cortar o permitir el flujo de un fluido, en nuestro caso agua (simulando el flujo del combustible).

3.2.3 Cámara

La cámara era uno de los componentes un tanto desafiantes pero el cual consideramos importante, ya que proporcionaba presencia y control sobre la situación del auto al dueño, que sin el mismo es encontrarse dependiendo de la reacción del sensor de movimiento el cuál puede generar una falsa alarma.

Utilizamos la ESP32-Cam la cual nos proporciona una cámara de buena calidad y un procesamiento integrado.

3.2.4 Solenoides

El solenoide era un componente que nos proporciona un interruptor mecánico fácil de usar y efectivo, lo consideramos para el mecanismo de bloqueo de puertas ya que era un beneficio útil para el dueño del auto poder controlar remotamente el bloqueo de las mismas.

3.2.5 Liberador de humo

Añadir un implementador de humo a la solución de nuestro proyecto lo consideramos algo “novedoso” y un desafío interesante para agregarle al vehículo. La idea del mismo era crear una cortina de humo dentro del vehículo para que sea imposible la conducción de él y de esta forma retrasar o impedir el hurto.

4. Planes iniciales

4.1 Ideas e implementaciones descartadas

Al principio, se tuvieron en cuenta muchos conceptos que terminaron siendo descartados, tanto por un tema de capacidad como por un tema de prioridades y tiempos.

A la hora de elegir qué implementaciones tendría el proyecto, tuvimos que priorizar las mismas para atacar lo más importante al principio, y descartar lo menos importante en el caso de que nos falte tiempo.

En este sentido, algunos elementos fueron descartados sobre el final por esa misma razón. El siguiente boceto representa la idea inicial del proyecto.

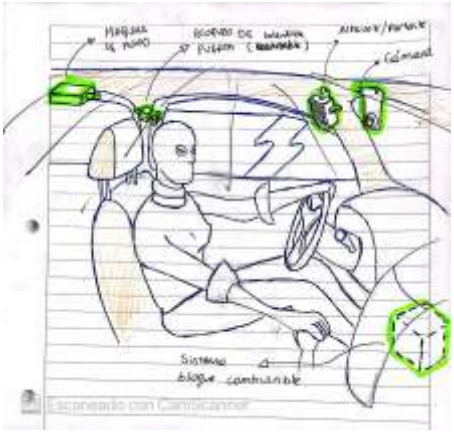


Fig. [1]: Dibujo concepto proyecto

Algunas de las implementaciones más relevantes serán mencionadas a continuación.

El parlante fue posicionado último en la lista de prioridades desde el principio. La idea con el mismo era poder comunicar tu voz dentro del auto mediante el celular u otro dispositivo para causarle al ladrón un estado de nerviosismo o confusión. Era una idea llamativa y divertida de agregar pero considerando el tiempo que lleva la implementación del mismo la decidimos descartar.

La pulsera también fue descartada por la misma razón. Su objetivo era ser utilizada por el dueño del auto, y en caso de que se sensara movimiento dentro del auto, la idea era que vibrara.

Otra idea que no llegó a ser elegida dentro de las fundamentales implementaciones fue la de tener dispositivo evidenciador ante choques cuando el vehículo se encontrase estacionado y sin el dueño cerca. Consiste de una serie de cámaras a los costados del vehículo y unos sensores de golpe (piezoeléctricos) que al sentir un golpe lo suficientemente fuerte se activaran las cámaras para tomar fotos y almacenarlas. De esta forma se tendría evidencia de la matrícula ante un choque el cual no se haya podido presenciar.

Esto no se llevó a cabo, además de no considerarse primordial, debido a el costo elevado de las placas con cámara.

Lo bueno de estos “fracasos” es que pudimos aprender a ser más realistas con nuestros objetivos en el campo de la ingeniería, viendo que nunca todo va a ir acorde a lo planeado.

4.2 Prototipos y pruebas de concepto

En el inicio del realizado de la solución, se plantearon diferentes prototipos a llevar a cabo, los cuales tenían como fin separar la solución en problemas más pequeños y controlables. Esta modalidad luego se va a ver adoptada en nuestra planificación.

Los diferentes prototipos que se definieron fueron los siguientes:

- Un sensor de movimiento integrado con una placa, la cuál estaría a cargo de enviar los datos de telemetría sobre el movimiento, recolectados por el sensor.

- Una cámara integrada a una placa para responder y realizar el envío de imágenes captadas.

- Un sistemas con solenoides y un relé para, mediante una señal de control encender los solenoides y trancar tanto las puertas como el flujo de combustible.

- Una placa con un dispensador de humo que permitiese liberar a pedido del usuario.

De ello, se realizaron varias pruebas de concepto. Por el lado físico se

discutió varias posibles realizaciones para los circuitos de alta potencia y como separar la placa del mismo, con lo que finalmente se llegó a la utilización del previamente mencionado relé.

A su vez se investigó sobre las capacidades y características del sensor de movimiento y de la cámara previo a su definición, ya que los mismos generaban dudas.

En específico, la utilización de una cámara lo consideramos un desafío ya que ninguno se encontraba familiarizado con cómo hacer que la misma captura imágenes, además del manejo y envío de las mismas. Tomamos la decisión de asumir este riesgo ya que desde nuestra visión, si lográbamos enviar la foto, procesar la imagen del lado de la nube y otros servicios no sería un problema.

Finalmente, el riesgo que conllevaba integrar un liberador de humo no tenía que ver principalmente con lo técnico, sino lo económico. Esto es debido a que luego de nuestra investigación inicial creímos que debíamos buscar una alternativa a las máquinas de humo, por más de que inicialmente no teníamos con claridad una alternativa decidimos confiar en nuestro “espíritu ingenieril” teniendo en cuenta este riesgo.

4.3 Planificación

Previo a comentar sobre la planificación, cabe definir algunos conceptos previos, relevantes para la explicación de la misma.

4.3.1 Metodologías ágiles

Se le conoce al desarrollo ágil de software o metodologías ágiles a un acercamiento distinto al manejo de proyectos y desarrollo de software, es una filosofía que supone una forma distinta de trabajar y de organizarse. Un equipo “ágil” entrega trabajo en pequeñas porciones que son más fáciles de digerir. Los requerimientos, planes y resultados son evaluados constantemente de manera que el equipo tenga un mecanismo de respuesta más natural.

4.3.2 Scrum

“Scrum” es un entorno de trabajo ágil para manejo de proyectos y es el más común utilizado por los desarrolladores de software a la hora de organizarse e iterar sobre el progreso del proyecto. Es el favorito principalmente debido al ambiente cambiante que resulta ser el desarrollo de software.

Con Scrum, un producto es construido a través de varias iteraciones a las cuales se le llama “Sprints”, estas desglosan grandes partes del proyecto en objetivos más pequeños y realizables a corto plazo. Se realizan en un período de tiempo definido y en él el equipo trabaja para completar una cantidad de tareas definidas.

Fig [2]: Ciclo del sprint (“Scrum: qué es, cómo funciona y por qué es excelente”, Atlassian)

4.3.3 El plan

Una vez mencionado lo anterior, pasaremos a explicar el plan de trabajo nuestro.

Para su planificación, debido a que ambos utilizamos en nuestros trabajos metodologías ágiles, decidimos dividir el proyecto en muchas “épicas” (conjunto de tareas) y la realización de sus diferentes partes en sprints.

En nuestro caso, las tareas definidas que planteamos a completar eran subtareas de una “épica”, cada épica definida era una implementación de las previamente mencionadas y la duración de las mismas rondan alrededor de las dos semanas cada una.

Durante estas dos semanas, se realizaría la implementación de cada una de las llamadas “features”, se harían pruebas de su implementación además de contar con tiempo extra en consideración de posibles imprevistos.

A su vez, se realizaría en paralelo la labor de documentar ciertos conocimientos adquiridos durante la correspondiente implementación, además de ingresar en una bitácora lo hecho durante la sesión.

De ser necesario más tiempo, consideramos dividir las tareas entre los miembros del equipo. Esto es debido a que en ciertos casos el desarrollo de una tarea es más eficaz si es realizada por una persona.

Fig. [3]: Diagrama de Gantt

4.3.4 Tecnologías aplicadas para el desarrollo ágil

Para facilitar el desarrollo ágil se hizo uso de varias tecnologías, principalmente utilizamos la herramienta Github.

Github es una plataforma para almacenar código que hace uso del sistema de control de versiones Git.

Git, a diferencia de otros sistemas de control de versiones, no se encuentra centralizado en un servidor. Sino que las versiones son replicadas o “clonadas” entre los dispositivos. Permitiendo así tener muchos backups y distintas versiones.

Con Git, estas son algunas de las cosas que se pueden realizar:

- Crear una “rama” en la que se pueden realizar ciertos cambios a código o probar nuevas cosas y volver al código viejo o intercambiarse a otra rama sin ningún problema.

- Experimentar sin consecuencias, si algo no funciona volver a atrás.

- Realizar un flujo de trabajo basado en “features” en el que se separan las nuevas implementaciones creando una rama para ellas.

Además de utilizar Github, decidimos adoptar un flujo de trabajo conocido como “Git Flow”, en el que los desarrolladores crean una rama de función separada de la rama principal, dónde se realiza todo el trabajo experimental, no se realiza una fusión con la rama principal del tronco hasta que la función está completa. Estas ramas de función son de mayor duración y para fusionarlas con la principal primero se realizaría un chequeo completo del código con motivos funcionales además de intenciones de mejorar el código para que sea más “limpio”.

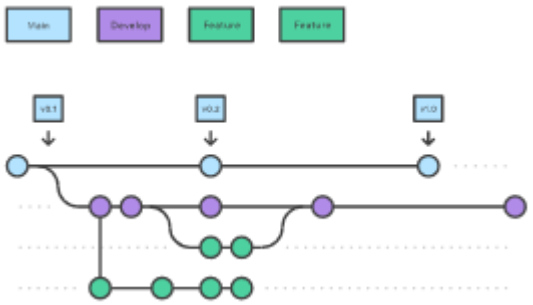


Fig [4]: Diagrama de ramas de trabajo (“Flujo de trabajo de Gitflow”, Atlassian)

También en el comienzo se dió la utilización de la herramienta conocida como “Notion”, una herramienta de manejo de proyectos visual para plantear y asignar tareas. La misma más adelante fue abandonada en su uso debido a que durante el desarrollo no nos fue de gran utilidad.

5. Implementaciones

En el siguiente fragmento se va a desarrollar en profundidad las soluciones realizadas para cada uno de los sprints, se van a mencionar los componentes incluidos, imprevistos, inconvenientes y el resultado final.

5.1 Thingsboard y PlatformIO

Como entornos iniciales que fueron replicados y utilizados durante todas las implementaciones, es relevante introducirlos previo a las soluciones de las partes de proyecto.

5.1.1 Thingsboard

Primero, como ya mencionamos anteriormente, Thingsboard es una plataforma “Open source” para IoT, hospedado en la nube. Con el mismo podemos realizar integraciones entre dispositivos para recibir datos de telemetría del mismo así como enviar llamados “RPC”.

“RPC” o “remote procedure call” es un protocolo de comunicación de software que un programa puede utilizar para pedir un servicio de un programa localizado en otra computadora. Este utiliza el modelo cliente-servidor, donde el programa que realiza el pedido es el cliente, y el servicio que provee el programa es el servidor.

Fig. [5]: Diagrama pedidos RPC (“Using RPC capabilities”, Thingsboard)

Thingsboard utiliza diversos protocolos de comunicación para lograr está conexión RPC, entre ellos nosotros utilizamos el llamado MQTT.

MQTT es un protocolo de red ligero, de publicación y subscripción lo que lo hace ideal para conectar dispositivos remotos con una huella de código pequeña y un uso mínimo de ancho de banda.

La forma en la que funciona la arquitectura de publicación y subscripción es, un dispositivo o servidor es el denominado “Broker” y otro es el “Cliente”. El Cliente se suscribe a un tópico del Broker, cuándo el Broker publica algo a ese tópico el Cliente recibe esa publicación. En ciertos casos el llamado puede ser bilateral, dejando un tiempo previo a un “timeout” en el que el dispositivo puede responder, invirtiendo así los roles.

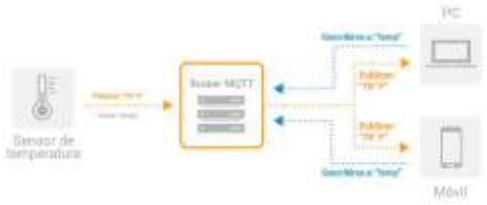


Fig. [6]: Diagrama comunicación MQTT (“MQTT: Conecte a dispositivos RedGps”, RedGps)

Por otro lado, Thingsboard también provee una interfaz para el propietario del servidor del mismo. En él se puede virtualizar diferentes dispositivos para realizar acciones con él dentro de la interfaz, en el que luego a través de un “token” de acceso vincular el dispositivo real.

En esta interfaz, las principales cosas que contiene son un “dashboard” dónde se tienen objetos diseñados para interactuar y observar, asemejándose a objetos reales tales como interruptores, leds, maps, entre otros.

En el mismo realizamos los ”widgets” para mostrar diferentes datos de los dispositivos además de tener interruptores virtuales para accionar ciertos actuadores.

Por último, el servicio cuenta con una sección dedicada a “cadena de reglas”, se le llama así debido a que contiene diversos bloques los cuales permiten replicar acciones que serían realizados a través de código y los mismos bloques son interconectables entre sí, creando así una serie de llamados en cascada o “cadena”.

Thingboard cuenta con otras secciones relevantes, que mencionaremos más adelante cuando sean utilizadas.

5.1.2 PlatformIO

La adhesión de la herramienta PlatformIO surgió a raíz de las intenciones de poder mantener una estructura de código más limpia. Ya que la programación en el IDE de Arduino suele ser más tediosa a la hora de utilizarse en conjunto con el control de versiones de Github, además de obligarnos a programar en distintos lugares los códigos de cada placa ya que sólo se puede tener un archivo “main.cpp” por proyecto.

Debido a lo dicho previamente, decidimos investigar si existía tal herramienta capaz de solucionar esto. Nuestro tutor nos comentó la existencia de esta extensión llamada PlatformIO, por lo que decidimos sumergirnos e indagar en la misma.

PlatformIO es, como indica su nombre, una plataforma colaborativa para desarrollo embebido. Cuenta con una galería de plataformas, frameworks y placas con la cuál es compatible y el mismo permite realizar desarrollo multiplataforma y multi-arquitectónico.

En sus distintas versiones, cuenta con una extensión para el IDE y editor de código Visual Studio Code, en ella se puede realizar un proyecto completamente desde el IDE para luego construirlo, subirlo, monitorearlo y depurarlo desde allí.

Un proyecto de PlatformIO cuenta con la siguiente estructura:

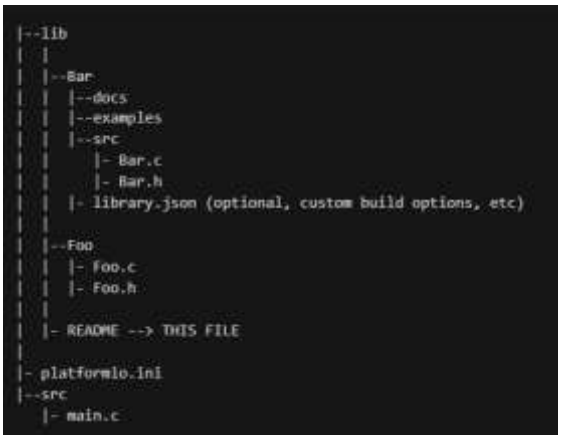


Fig. [7]: Estructura proyecto platformio

- Una carpeta “src” dónde va el archivo main.cpp entre otros .cpp, archivos de C + + que contienen la lógica.
- Una carpeta “lib” para incluir bibliotecas privadas.
- Una carpeta “include” dónde van los archivos .h del proyecto.
- Una carpeta “tests” dónde van archivos para realizar testing unitario.
- Un archivo platformio.ini, dónde los entornos de platformio se configuran.

A la hora de realizar un código para una placa, incluímos un archivo para realizar toda la lógica, además de agregar las bibliotecas necesarias. Creamos un entorno específico para esa placa lo cuál nos permite separar los archivos necesarios de cada implementación para que sean construidos y subidos a la placa.

Lo más importante de ello, nos permite tener una estructura de mono-repositorio en dónde el código de todas las placas se encuentra en un lugar y gracias a una ventana de scripts podemos tener un despliegue fácil de cualquiera de los códigos de las placas.

Para más información sobre PlatformIO, observar el documento incluido en el código dentro de la carpeta docs con una guía para iniciarse con la plataforma y cómo crear un proyecto.

5.2 Movimiento y localización

La implementación para el sensor de movimiento fue la primera de las mismas, ya que, a raíz de la investigación inicial, resolvimos que era la más directa.

Comenzamos adquiriendo el sensor PIR, y probando su funcionalidad con un código muy simple para asegurarnos que el producto vino en buenas condiciones.

El siguiente paso fue revisar el código de ejemplo que los tutores habían realizado para demostrar cómo realizar la conexión WiFi y MQTT inicial, además de suscribirse a los tópicos correspondientes.

También tuvimos una lección del proceso para vincular un dispositivo con su parte virtual en Thingsboard mediante un token de acceso único.

De allí resolvimos realizar un código similar que seguía los mismos pasos, la placa se conecta a la red WiFi, se vincula por MQTT a los tópicos correspondientes en Thingsboard y mantiene subrutinas para enviar datos de telemetría además de chequear la conexión. Para enviar los datos se utiliza la biblioteca “ArduinoJSON” en la que se formatea los datos que queremos enviar para que se mande como un objeto JSON.

“JSON” o “Javascript object notation” es un formato de texto sencillo y normalizado para el envío de datos en la web especialmente para peticiones HTTP. Mantiene una estructura similar a la de un objeto, conteniendo un par clave-valor que puede ser de tipo string, number, boolean u otro objeto.

Fig. [8]: Ejemplo formato JSON

Adelantándose a los porvenires posibles en dónde este proceso se iba a replicar varias veces, observamos que, por la naturaleza de los archivos “main” hacia las placas, el código iba tener que ser necesariamente redundante. Sin embargo, ciertas constantes y algunos valores no tenían porqué serlo. Así que tomamos nuestra primera decisión hacia un código más limpio y decidimos armar un archivo .h que tendría en él los valores de los tokens además de configuraciones de red entre otros.

Una vez finalizado el código, configuramos Thingsboard para poder recibir los datos que manda el sensor de movimiento. De allí añadimos un dispositivo del cuál su token de acceso fue agregado al archivo de configuración y creamos nuestro primer widget, un LED que se enciende cuándo recibe un dato de movimiento. Ésta lógica es resuelta a través de la cadena de reglas, la cadena cuenta con una versión predeterminada con los bloques básicos, que en nuestro caso son los que necesitamos para ésta implementación. Simplemente se requiere un filtro para mensajes de tipo telemetría y un bloque para enviar los atributos al dashboard.

No surgieron percances durante está implementación, lo más duradero fue comprender cómo enviar los mensajes

desde la placa y cómo recibirlos, pero en su totalidad llevó menos tiempo de lo esperado.

Ya que no nos había llevado tanto tiempo, además de sentir que no estábamos sacándole todo el provecho a la placa, decidimos agregar una funcionalidad más a la misma, añadiendo así la parte de localización.

La funcionalidad de localización la teníamos pensada desde el principio como un agregado, aunque lo encontrábamos lejos de ser implementado, principalmente porque no contábamos con un módulo GPS como para obtener la localización.

Fue en ese momento, que a raíz del tiempo ganado de la implementación de movimiento, que nos topamos con una alternativa ideal. Encontramos una API de Google que, pasándole como datos las direcciones MAC y sus intensidades de señal, nos devuelve la latitud y longitud con cierto márgen de error. Antes de seguir, vamos a explicar que significa API y direcciones MAC.

API significa “interfaz de programación de aplicaciones”. Es un mecanismo que permite a dos servicios o componentes conectarse entre sí a través de definiciones de métodos. Un servicio ofrece y expone una API para que otros utilicen los datos que ellos proveen, sea de forma directa o detrás de escena mediante una página. El proceso más común es un intercambio en el que el “cliente” realiza un pedido, enviando cierta información o completando un procedimiento, y el servidor responde con información a ese pedido.

El tipo de API más popular de encontrar en la web es REST, que significa transferencia de estado representacional.

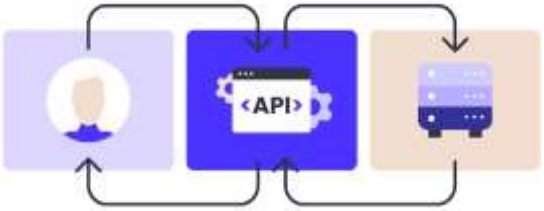


Fig. [9]: Ilustración API (“¿Qué es una API y para qué sirve?”, B2Chat)

En el mismo se define un conjunto de funciones a las que se les otorga la sigla de CRUD, que en órden conforman las palabras CREATE, READ, UPDATE y DELETE. Son las funciones básicas en las que se desglosa toda operación y que los clientes pueden utilizar para acceder a los datos del servidor así como modificarlos. Se intercambia datos mediante el protocolo HTTP.

Continuando, la dirección MAC (Media Access Control) es un atributo de 48 bits en hexadecimal que se utiliza como identificador único de un dispositivo electrónico en una red. Es también conocida como la dirección física.

Retomando entonces con el descubrimiento, decidimos utilizar esta API para obtener las coordenadas sin la necesidad de comprar un GPS. Para ello tuvimos que profundizar más en la utilización de Thingsboard. Este cuenta con un bloque en la cadena de reglas para realizar un llamado a una API y retornar con los valores devueltos.

Para ello requerimos de la placa para enviarle a Thingsboard los datos de las direcciones MAC circundantes. Por lo cual añadimos un bloque generador de llamados RPC que se realizaría cada 20 segundos y lo enviamos como pedido a la placa. De allí creamos una función en la placa que respondería al método de conseguir las direcciones MAC proveniente del pedido RPC. Esta función es bastante directa, utiliza la biblioteca

WiFi previamente en uso para la conexión a internet y con ella realiza un escaneo de las redes para ser almacenados en una variable. Con esos datos los formateamos en un JSON y respondemos al RPC con esa información.

Una vez respondido el RPC, los datos siguen en la cadena de reglas donde con un bloque de script se arma el objeto JSON a mandar como pedido a la API, para que luego este responda con la información.

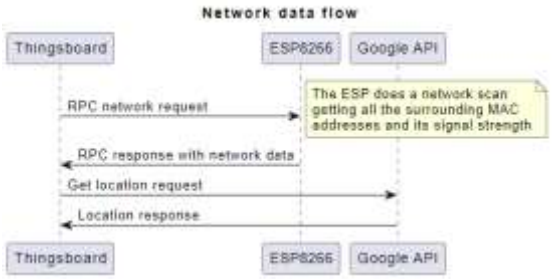


Fig. [10]: Diagrama de flujo sobre la información de la red

A partir de la respuesta de la API, separamos los valores devueltos con otros bloques y los asignamos a atributos del servidor creados para ellos, latitud y longitud.

Para poder visualizar la posición, agregamos un widget de mapa, que vinculado con los datos de latitud y longitud te muestra en el mapa la posición y con ello finalizamos la implementación.

5.3 Válvula solenoidal y tranca para puerta con solenoides

Estos elementos fueron categorizados en una misma sección porque esencialmente son elementos muy parecidos.

Físicamente, ambos componentes tratan de una bobina que se activa y genera cierta reacción en cada elemento. En el caso de los solenoides, la activación de la bobina implica un pequeño impulso de un cilindro de metal que pasa por dentro de la misma. En el caso de la válvula, consiste de una puerta que impide el paso de líquidos por la misma. A nivel de circuito, ambos componentes utilizan un relé que permite controlar la activación de los dos con una señal de control.

Digitalmente, el código de ambos elementos funciona de manera muy similar; prácticamente es una señal que llega en un paquete JSON bajo el parámetro “method” que le indica cada uno en el momento que se tenga que activar. El código procesa la petición y activa los elementos.

Con respecto a su funcionalidad, ambos componentes son muy distintos. Por un lado la válvula es de las features más importantes del proyecto. Su función principal es cortar el combustible antes de que el delincuente pueda prender el auto. Por otro lado, los solenoides no cumplen un rol de la misma índole o incluso de la misma prioridad. Su función es secundaria, y es básicamente permitirle al cliente poder trancar el auto en caso de no haberlo trancado.

Con respecto al funcionamiento, el de la válvula es el siguiente. Al activarse el sensor de movimiento y percibir un rostro en la cámara, Thingsboard se encarga de enviarle un mensaje a la placa de la válvula en forma de parámetro. Este parámetro es un “method” llamado “cutEngine”. El mismo le indica, por internet, a la placa de la válvula que tiene que activar la misma. Por está razón, la placa le envía una señal mediante un pin a la válvula, indicando que se active. El componente no tiene entrada de control, por lo que tuvimos que conseguir un relé, cuya funcionalidad es permitir el paso de corriente si la entrada

de control recibe voltaje. De lo contrario, no permite el paso. Es de está forma que la válvula se activa únicamente si la placa envía una señal. Resumidamente, el proceso es:

1. El sensor de movimiento se activa.
2. La cámara procesa un reconocimiento facial.
3. La válvula corta el combustible.

Regresando a los solenoides, los mismos tienen un proceso de activación bastante más simple. Para la activación de los mismos, lo único que tiene que pasar es que el cliente los active mediante la aplicación de thingsboard. No cuenta con activación automática. Por el mismo proceso que la válvula, cuando el cliente aprieta un boton de activacion, thingsboard se comunica con la placa de los solenoides y les manda un “method” llamado “doorTrigger”, que envía una señal de activación por un pin que está conectado al relé. Una vez que recibe la señal, el relé permite el paso de corriente para los solenoides y se activan.

Inicialmente, teníamos en mente activar el solenoide para trancar el auto, y desactivarlo para trancarlo. Luego de algunas pruebas la realidad y la experimentación nos enseñó que esto no es posible. Si bien en teoría la bobina tiene la capacidad de almacenar energía, en la práctica está capacidad no es ilimitada. Lastimosamente la misma se disipa en calor, y si está activada un tiempo largo, el componente no lo soporta y se quema. Esto no le sirve al proyecto dado que nosotros pretendemos que el auto se quede trancado una vez que se activan los solenoides, y si los mismos no pueden estar activados no podemos cumplir la feature.

Discutimos en el equipo distintas soluciones complejas, pero la conclusión fue que no podíamos implementar ninguna ya que la prioridad del proyecto era otra, y todas esas soluciones planteadas iban a demorar mucho tiempo de investigación e implementación. Sin embargo, nos pareció buena idea dejar la solución conceptualmente planteada dentro de la maqueta, de esta forma podríamos investigar para implementarlo en futuras actualizaciones del producto.

En este sentido, con fines conceptuales para esta maqueta, el solenoide tuvo que colocarse de tal manera que la puerta siempre esté trancada. Entonces al recibir la señal, la tranca se destraba en lugar de trabarse. Por lo que la puerta siempre está trancada y se destranca cuando recibe la señal, y esto sirve para abrirla. Una vez que se activan los solenoides, la placa espera 2 segundos, y luego envía una señal sin voltaje para que se desactiven. Resumidamente:

1. El cliente aprieta el botón de activación.
2. Los solenoides se activan y destrancan la puerta.
3. Luego de 2 segundos, automáticamente el solenoide se vuelve a trancar.

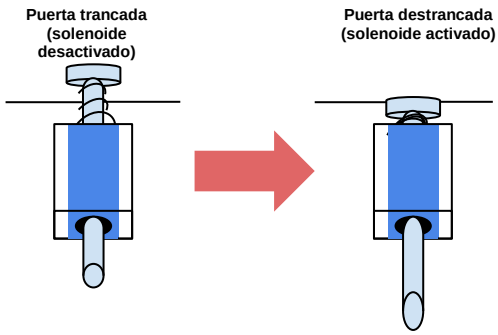


Fig. [11]: funcionamiento solenoides.

Esta idea fue planteada inicialmente en el anteproyecto, y su fin principal era aterrorizar al delincuente una vez que se diera cuenta que no podía salir

del vehículo gracias a los solenoides (controlados por el usuario). Luego, en la implementación de la misma, nos dimos cuenta que está idea era inmadura y no tenía mucho sentido. Su esencia era aterrorizar en lugar de proteger, lo cual va en contra de los fundamentos de Guardian Engine. En concreto, la idea no correspondía con el objetivo del proyecto: proteger la seguridad del auto. Incluso, era opuesta a la misma. En el caso de un robo, el interés del cliente es ahuyentar al delincuente para prevenir el mismo. Si Guardian Engine impide que el delincuente huya, estamos logrando lo contrario de lo que justamente queremos conseguir.

Es en este sentido que incluso consideramos descartar la idea. Sin embargo, vimos la feature de trancar como un buen agregado a la seguridad. Teniendo autos para manejar nosotros, nos dimos cuenta que muchas veces nos sucede que nos olvidamos de trancar el vehículo o incluso no recordamos haberlo trancado. Nos pareció una muy buena solución a una necesidad el hecho de poder trancar el auto a muchos kilómetros de distancia, y es por esto que decidimos agregar, al menos conceptualmente, la feature de los solenoides.

5.4 Cámara y detección de personas

Comenzando con la implementación de la cámara, que adelantando a lo que se va a mencionar en la brevedad, fue la más compleja de entender y hacer funcionar.

Iniciamos investigando sobre el componente de la cámara además de cómo implementar sus funcionalidades. Con ello ideamos un plan sobre cómo integrar la cámara con Thingsboard.

Además, en paralelo, empezamos a cuestionarnos si el enfoque que estábamos tomando con otros componentes era el mejor. Esto debido a que algunos de los actuadores hasta el momento se encontraban entre depender del sensor de movimiento o de la activación manual del usuario.

Lo último era lo menos deseado ya que nuestras defensas perderían sentido alguno si el usuario debía estar pendiente constantemente del estado de su vehículo, pero por el otro lado el automatismo que provee el sensor de movimiento no es muy fiable por sí sólo al poder ser propenso a falsas alarmas.

Fue a raíz de esta situación que hallamos una API de reconocimiento facial y de cuerpos humanoides llamada Face++ que nos permitió resolver esta problemática. Si realizamos el envío de la imágen a la API para corroborar la presencia de un intruso el automatismo pasaría a ser inteligente y con mayor veracidad.

La idea era concisa, luego de recibir una señal de detección de movimiento, realizaríamos un llamado RPC a la cámara para que tome una foto, esa foto sería devuelta como respuesta para luego pasar por la cadena de reglas y ser enviada como parámetro al llamado de la API. Una vez retornado la respuesta de la API, en caso de haber detectado un cuerpo, realizaría los automatismos con la válvula solenoidal.

Encontramos varios códigos de prueba cómo el de ejemplo que se encuentra en el IDE de Arduino, que permite levantar un servidor HTTP para transmitir las imágenes captadas por la cámara. Además de otro código en el que decidimos basarnos.

Este último contenía una función para tomar una foto y, a partir de una biblioteca realizada por el usuario, codificar la imágen en base64. Base64 es un grupo de esquemas de codificación de binario a texto el cual representa sus datos con una cadena ASCII, es además uno de los formatos de imágen que la API nos permitía enviarle como dato. Comenzamos intentando con una versión modificada de este código pero no daba resultados por lo que volvimos al ejemplo original.

A partir de este código, chequeamos la funcionalidad de la cámara, allí comprobamos que debido a varias razones no estábamos transmitiendo. Una vez resuelto que requerimos de una tarjeta SD y que todo estaba correcto tuvimos éxito en el código de ejemplo, de ahí no tardó mucho en funcionar la primera iteración del código.

Fue en este punto que surgió nuestro primer bloqueo con la cámara. El llamado se hacía correctamente y se mandaba en la respuesta el string de base64 de la imágen, pero el mismo figuraba con error en la cadena de bloques.

Había varios posibles causantes de esta situación, principalmente creemos que tenía que ver con el tamaño del string enviado ya que era considerablemente más largo que los otros cuerpos de respuesta de RPC. Además, el string de base64 variaba en tamaño ya que dependía de la imágen y los colores involucrados, esto generaba que para enviarlo debíamos comenzar la respuesta RPC, enviar los datos por baches y finalmente terminarla.

Luego de varios intentos de solucionar esto, terminamos abandonando este acercamiento dado que no parecía haber solución fácil y, por más que no era lo deseado, optamos por realizar el llamado a la API desde la placa con una solicitud HTTP para luego enviar la respuesta de detección de persona.

Este segundo acercamiento trajo también sus problemas, ya que la API por motivos de seguridad sólo permitía realizar llamados con HTTPS. Para lograr esto incluimos un certificado en la configuración, pero generó conflictos en la placa con errores relacionados al espacio de memoria. Luego de intentos de depurar la situación y consultar en foros abandonamos esta idea.

Finalmente, retomamos el enfoque de tener servicios en la nube y en la web como principales actores, fue entonces que investigando un poco ideamos un plan. Subir la imágen a un servidor FTP para que, en lugar de enviar la imágen por MQTT teniendo en cuenta el tamaño extenso que tenían los datos, pasar cómo parámetro el URL dónde se encontraba esta imágen ya que es un parámetro aceptado por la API.

FTP o “file transfer protocol” es un protocolo estándar de red que se utiliza para el intercambio de archivos de forma segura sobre una red TCP/IP. El mismo utiliza el modelo cliente-servidor

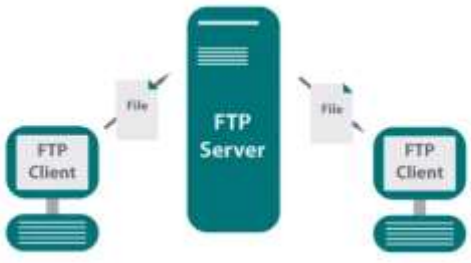


Fig. [12]: Modelo comunicación FTP (“What is FTP”, Medium)

Para ello se necesitaba un servidor FTP y un servidor web, investigando encontramos un proveedor llamado Byethost que brinda de forma gratuita varios servicios, entre ellos los requeridos.

Por el lado del código, incluimos una biblioteca para el protocolo FTP, en dónde se iniciaba la conexión de la misma forma que se realiza con WiFi y MQTT y agregamos que al momento de tomar la imágen se enviase al servidor anidando como URL el parámetro de id con el que se realizó el pedido RPC

Habíamos resuelto este inconveniente y finalmente no habían errores relacionados a la respuesta de la placa en la cadena de reglas, pero de ello surgió un último problema. A la API de reconocimiento no le gustaba el origen del URL ya que la conexión no era HTTPS, por lo que no realizaba el pedido. Abrumados nos hicimos a un lado con ésta API pero de ella surgió una opción mejor.

Hallamos una API de Google llamada Cloud Vision API que realizaba las mismas acciones que Face++ y al utilizarla finalmente todo se comprobó funcional el flujo completo.

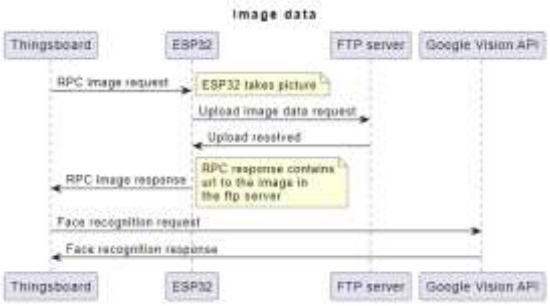


Fig. [13]: Diagrama de flujo del pedido y reconocimiento de imágen

Fué así como obtuvimos una versión funcional de flujo completo, a partir de la respuesta de la API el flujo continúa para activar el bloqueo de flujo de combustible.

Para complementar agregamos a su vez interfaces para el usuario poder estar pendiente ya que disponíamos de la imágen subida a una página web con la que interactuar. Por eso agregamos que se subiese el último URL como atributo de servidor para ser utilizado en los widgets de HTML y de mostrar atributos.

Configuramos el JSON del widget HTML para aceptar atributos y vincularlo con el URL para ser utilizado como parámetro, pero este se mostró como imágen inaccesible, luego de intentar con otras imágenes filtramos la opción de que fuera un error de la configuración realizada en el widget.

Lamentablemente dado la naturaleza del servidor y que el URL no accede directamente a la imágen era imposible mostrar la imágen un tag de imágen. Más allá de eso mantenemos en el otro widget el URL de la última imágen subida a la cuál se puede acceder pegando el URL en un navegador.

Fig [14]: Imágen de widgets de imágen

5.5 Dispensador de humo

Por último tenemos, la implementación más ingeniosa del proyecto por su carácter de solución barata y eficaz para prototipo.

Desde el comienzo sospechamos que una máquina de humo sería un elemento muy caro de implementar en el proyecto, por lo cuál dependemos de

encontrar alternativas para hallar cómo liberar humo.

Luego de consultar alternativas con conocidos, caímos en una solución bastante interesante, utilizar un vaporizador de humo descartable como sustituto.

El vaporizador contiene una esponja con líquido y una resistencia dentro, a sus extremos tiene aberturas para aire. Cuando una persona aspira por la boquilla se genera un flujo de aire que es detectado por un sensor para enviar una señal de control y cerrar el circuito con la batería, esto calienta la resistencia generando así el humo.

Sin embargo, el vaporizador sólo libera humo con flujo de aire, el cuál no podía originar de una persona. Decidimos entonces fusionar este vaporizador con un ventilador pequeño. Ambos serán encendidos con una señal de control a la vez brindándonos así un dispensador de humo.

Conseguimos entonces un dispensador de humo desechable y nos asesoramos en cómo desarmarlo y recargar tanto la batería como el líquido. Una vez desarmado y separado de la batería, observamos la potencia que utilizaba y comenzamos a probarlo con una fuente externa.

Con ello, realizamos varias iteraciones del circuito, principalmente debido a que el objetivo era poder alimentar tanto el ventilador de 5 volt como la resistencia que hasta el momento se le suministraba 1.85 watt de potencia. En varios de estos intentos terminamos con el mismo resultado, la resistencia acapara toda la potencia dejando al ventilador sin poder moverse, todo esto utilizando un transistor como llave para ambos circuitos.

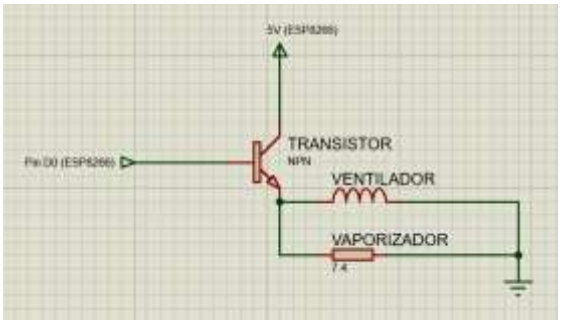


Fig [15]: Primer iteración circuito

Decidimos entonces separar los circuitos con dos transistores y alimentar al vaporizador con el pin de 3.3 volt en su lugar.

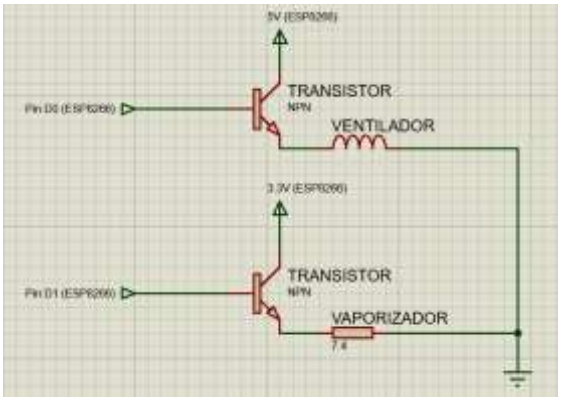


Fig [16]: Segunda iteración circuito

Esta solución aparentó funcionar, más adelante al montar todo generó problemas similares con el ventilador ya que no tenía la fuerza para arrancar pero sí para continuar una vez hecho un empujón.

Por ello decidimos buscar un reemplazo de batería con la qué alimentarlo, buscando reutilizar entre juguetes viejos encontramos un perfecto candidato con el cuál funcionó a un mejor nivel. El problema surgió con el transistor que se empezó a calentar por lo cuál, ya que contamos con un relé decidimos utilizarlo con éste también.

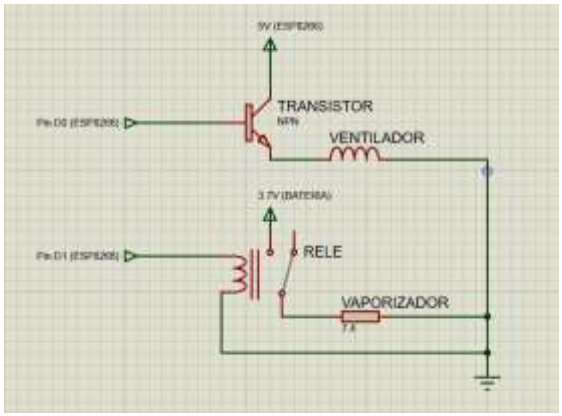


Fig [17]: Última iteración circuito

6. Maqueta y ensamblaje

Para la maqueta en principio teníamos varias ideas, realizar una maqueta de dimensiones grandes o realizar la misma a tamaño real lo que sería la parte de adelante del auto. Lo que era seguro es que debido al tamaño de los componentes, realizar una maqueta pequeña carecía de sentido para nosotros.

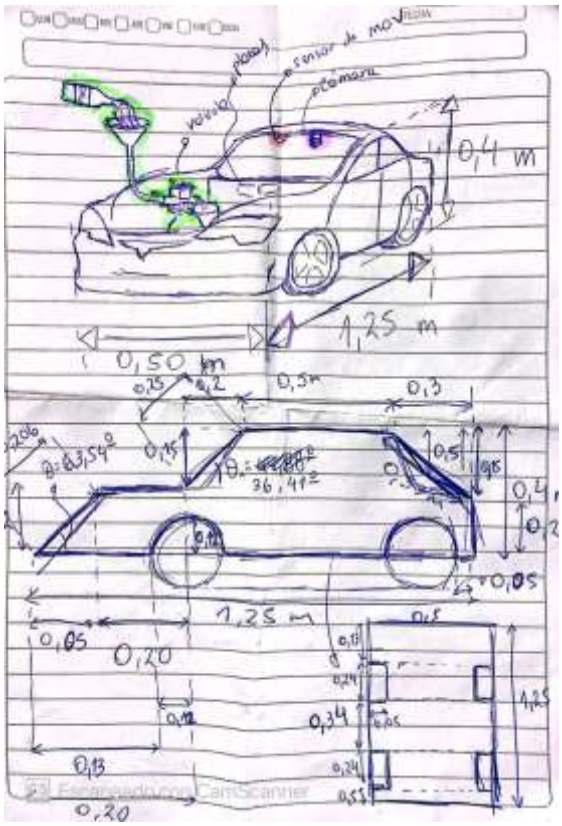


Fig [18]: Esquema maqueta

A partir de ello, se realizó un esquema con medidas a seguir para la construcción, se eligió hacerla de cartón debido a que es un material bastante manejable y se comenzó a construir.


Fig [19]: Maqueta

Luego se comenzó con el ensamblaje de los circuitos, se compró un puerto para 4 USBs en dónde centralizar la conexión de las placas y se comenzó a montar todo.

Finalmente se decidió pintarlo para darle mayor atractivo a la maqueta para así completarla.


Fig [20]: Montaje en maqueta


Fig [21]: Maqueta finalizada

7. Conclusiones y mejoras a futuro

Ambos creemos que fue una experiencia muy enriquecedora que pudo demostrar un avance en nuestro aprendizaje y “problem solving” tanto a raíz de la evolución por técnicas aprendidas en el ambiente laboral como en la carrera, sin mencionar a su vez organización y estimación de tiempos.

Fue un buen punto de comparación con la dinámica de investigación y conocimiento previo respecto a la materia Proyecto Integrador 1 ya que se notó una gran diferencia con énfasis en el comienzo.

Sobre los componentes, nos impresionó lo que se puede llegar a realizar con las placas ESP8266 y ESP32, ambos a un precio relativamente bajo con muchas posibilidades. En especial la cámara probó ser muy eficiente y de muy buena calidad con imágenes buenas además de un montaje fácil del lado de hardware.

En ciertos puntos consideramos a su vez que nos complicamos con lo que buscábamos pero también era parte del desafío. Por suerte supimos resolver y llegar a lo que buscábamos o una aproximación muy cercana.

Por otro lado, nos encontramos sorprendidos al estar por debajo del presupuesto con respecto a lo que compramos, fueron compras bastante eficientes y debido al dinero de sobra en el caso del ventilador nos dió márgen para comprar diferentes y poder buscar el más óptimo.

Thingsboard por su lado tiene muchas cosas buenas por todo lo que nos permitió realizar, más allá de eso también nos generó muchos problemas y complicaciones que quizás en una versión paga no ocurriría lo mismo así cómo en otra solución de la nube.

Como mejoras que pensamos realizar para un próximo prototipo, además de los agregados, podría ser en específico con las trancas del auto y el lanzador de humo. Esto se debe a que como muestra de prototipo son buenos ejemplos pero no para una versión final.

El sistema de las trancas podría ser modificado para que en vez su uso sea como un switch, esto se podría lograr consiguiendo un tipo distinto de solenoide o cerradura magnética así como realizando un mecanismo físico a integrar.

Por el lado del humo conseguir o realizar una versión más potente tanto con el ventilador como el vaporizador para mayor cantidad de humo.

A su vez, realizar carcasas para los componentes y sustituir breadboards por un tema de estética.

Como producto final, tenemos algunas implicancias que consideramos si fuésemos a vender el producto. Principalmente el servicio de la nube que utilizaríamos sería Microsoft Azure o Amazon Web Services. Por otro lado intentaremos realizar un producto que sea fácil de integrar sin un montaje de un técnico involucrado.

8. Anexo

8.1 Link a código en GitHub

https://github.com/Conectando-Cosas-2022/GuardianEngine

8.2 Link a página web y bitácora

https://guardianengine.bitrix24.shop/checkout/#

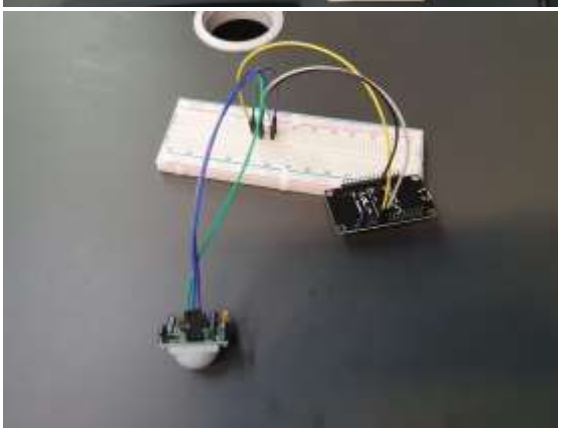
8.3 Link al vídeo

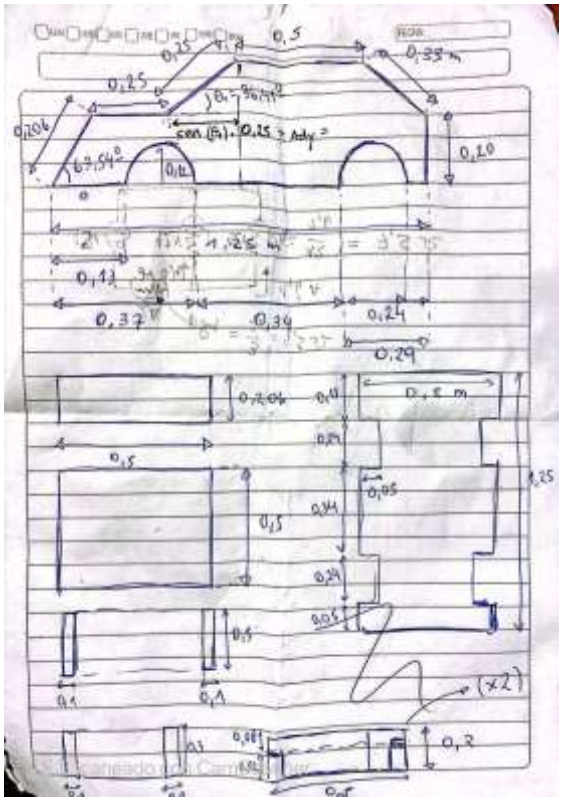
https://youtu.be/9OApBCnJWGA

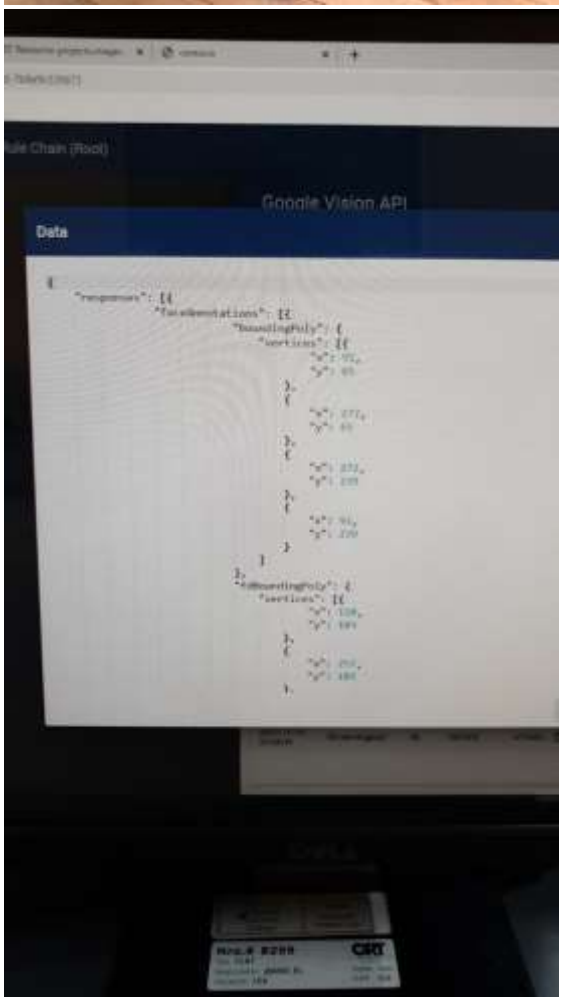
8.4 Dashboard

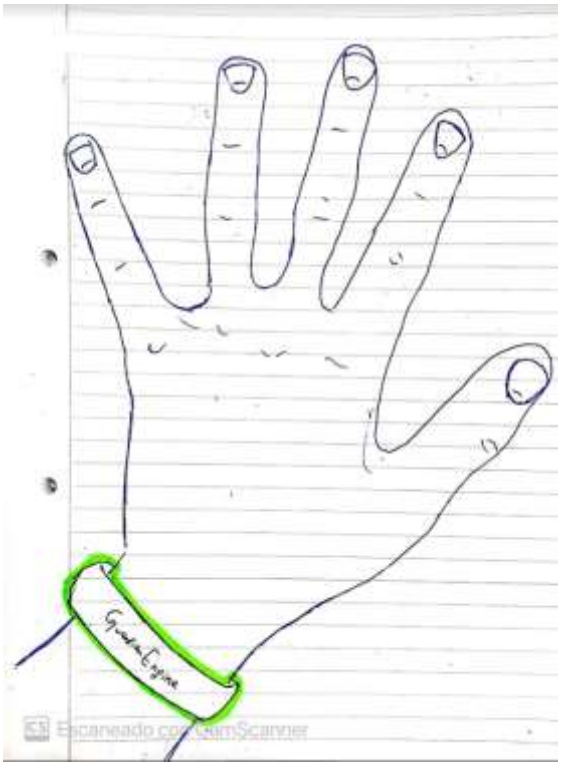
8.5 Poster

8.6 Imágenes









9. Referencias

- Body Detection - Face++ Cognitive Services (faceplusplus.com) - API Face++
- Location Tracker With NodeMCU ESP8266 : 10 Steps - Instructables - Cómo obtener localización con NodeMCU ESP8266
- ThingsBoard Map widget configuration guide. Part 1 - YouTube - Guia widget mapa
- https://www.instructables.com/A-Raspberry-Pi-Powered-Junk-Drum-Machine/ - Circuito Solenoide
- (9) How to Make Powerful Air Blower using Computer Fan - YouTube - Ventilador potente

- https://www.youtube.com/watch?v=9wVSzqkbEtk - PIR sensor tutorial
- https://elosciloscopio.com/hc-sr501-sensor-de-movimiento-pir-para-arduino-esp8266-y-esp32/ - circuito del sensor con la placa
- https://www.youtube.com/watch?v=QJc2ZUeBpn8
- https://github.com/fustyles/Arduino/blob/master/ESP32-CAM_MQTT/ESP32-CAM_MQTT.ino - codigo camara esp32
- https://github.com/thingsboard/thingsboard/issues/325
- https://github.com/thingsboard/thingsboard/issues/6567
- https://github.com/thingsboard/thingsboard/blob/master/application/src/main/data/json/system/widget_bundles/input_widgets.json
- https://groups.google.com/g/thingsboard/c/UUCMAG7-cWg?pli=1
- https://www.sarandi690.com.uy/2021/01/12/en-uruguay-se-roban-42-vehiculos-por-dia-el-portonazo-es-la-modalidad-mas-utilizada/
- https://www.electronica.uy/robotica/tarjetas-accesorias-de-desarrollo/m%C3%B3dulo-esp-32-wifi-bluetooth-con-c%C3%A1mara-detalle.html
- https://www.robotec.com.uy/productos/TX538
- https://www.electronica.uy/robotica/modulos-comunicacion/m%C3%B3dulo-nodemcu-v3-esp8266-wifi-detalle.html
- https://www.electronica.uy/robotica/modulos-control-potencia/m%C3%B3dulo-relay-1-canal-5v-7675-detalle.html
- https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp_http_client.html
- https://console.cloud.google.com/google/maps-apis/overview