Anexo A

Acrónimos

- **API:** Application Programming Interface
- **DEV:** Developer
- **IDE:** Integrated Development Environment
- **EC2:** Elastic Compute Cloud
- **DNS:** Domain Name Server
- **LTS:** Long Term Support
- **IP:** Internet Protocol
- **IoT:** Internet of Things
- **BTC:** Bitcoin
- **MTProto:** Mobile Transport Protocol
- **DB:** Data Base
- **LED:** Light Emitting Diode
- **FTP:** File Transfer Protocol
- **JSON:** JavaScript Object Notation

- **FIFO:** First In First Out
- **OAuth:** Open Authorization
- **URL:** Uniform Recurse Locator
- **UML:** Unified Modeling Language
- **AWS:** Amazon Web Services
- **SSH:** Secure SHell
- **HTTP:** HyperText Transfer Protocol
- **HTTPs:** HyperText Transfer Protocol Secure
- **MPEG:** Moving Picture Experts Group
- **VPS:** Virtual Private Server
- **CPU:** Central Processing Unit
- **ID:** Identifier
- **REQ:** Request
- **UTF-8:** 8-bit Unicode Transformation Format
- **PIP:** Python Package Index
- **SSL:** Secure Sockets Layer
- **HTML:** HyperText Markup Language
- **AAC:** Advanced Audio Coding
- **ADTS:** Audio Data Transport Stream
- **M3U8:** M3U utf-8 codified
- **ASCII:** American Standard Code for Information Interchange
- **RE:** Regular Expresions

- **E2E:** End to End
- **MAC:** Message Authentication Code
- **SHA-1:** Secure Hash Algoritm
- **PLN:** Procesado del Lenguaje Natural

Anexo B

Guía de instalación bot

Para poder utilizar la aplicación de gestión de la cámara de vigilancia es necesario instalar y configurar diferentes módulos. (si se desea acceder al bot ya implementado consultar paso 12). Los requisitos para poder correr la aplicación y que se dan por instalados son:
+ Tener una máquina con una distribución Linux.
+ Tener instalado intérprete Python 2.7 o superior.
+ Tener una interfaz de red pública accesible con port 8443 abierto.
+ Tener configurado el software de control git.

Este tutorial está orientado para la instalación en una máquina Ubuntu 16.04 LTS:

- 1. (opcional) Instalar Dynamic Update Client No-IP
  - wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz
  - tar xzf noip-duc-linux.tar.gz
  - make install
- 2. Instalación de la API Bot de Telegram :
  - git clone https://github.com/eternnoir/pyTelegramBotAPI.git

- 3. Configurar cuenta de desarrollador en Netatmo, y desde el menú CREATE YOR APP y configurar los campos según interés. Asegurar que el campo read camera esté activo. o
- 4. Descargar la traducción a Python de Netatmo DEV API
  - git@github.com:philippelt/netatmo-api-python.git
- 5. Instalar la librería de conversión ffmpeg3, la versión 2 no permite la conversión de formatos m3u8:
  - sudo add-apt-repository ppa:jonathonf/ffmpeg-3
  - sudo apt update && sudo apt install ffmpeg libav-tools x264 x265
- 6. Crear certificados openssl
  - openssl genrsa -out webhook_pkey.pem 2048
  - openssl req -new -x509 -days 3650 -key webhook_pkey.pem ?out webhook_cert.pem
- 7. Dar de alta un bot en la aplicación Telegram. Iniciar conversación con @botFather y enviar la petición /newbot. Enviar nombre del bot y nombre de usuario. El contacto devuelve el API token para su configuración.
- 8. A partir de este momento el bot es accesible desde cualquier aplicación cliente Telegram. Si se desea acceder al bot ya implementado y operativo que gestiona la cámara del departamento de Telemática se puede acceder directamente a través de https://telegram.me/netwelcomebot

Anexo C

MTProto

En este anexo se va a profundizar en el protocolo MTProto ampliando la información sobre su funcionamiento y su implementación en Telegram. Es en definitiva el protocolo que usa el bot desarrollado para comunicarse con el usuario.

Telegram implementa un protocolo propio, MTProto, que transmite los mensajes de forma segura entre nuestro móvil y el servidor. Incluso permite crear chats seguros entre dos clientes, con cifrado e2e para que ni siquiera los propios servidores puedan acceder a la información.

Cada vez que se realiza una comunicación se produce cifrado cliente-servidor, cada mensaje conforma cuatro elementos:

+ **Salt:** 64 bits aleatorios, que se cambian en cada petición al servidor para evitar ataques de reply.

+ **Session_id**: 64 bits generados por el cliente, para identificar su instancia de sesión.

+ **Payload:** Además de contener el contenido del mensaje, incluye la fecha, longitud y número de secuencia del mensaje.

+ **Padding:** bytes de relleno.

Una vez conformado este mensaje se utiliza un MAC (Message Authentication Code). Este código se obtiene a partir del mensaje, y es

(casi) único. Si el mensaje cambia, aunque sólo sea en una tilde, el MAC será distinto, es similar a las funciones hash.

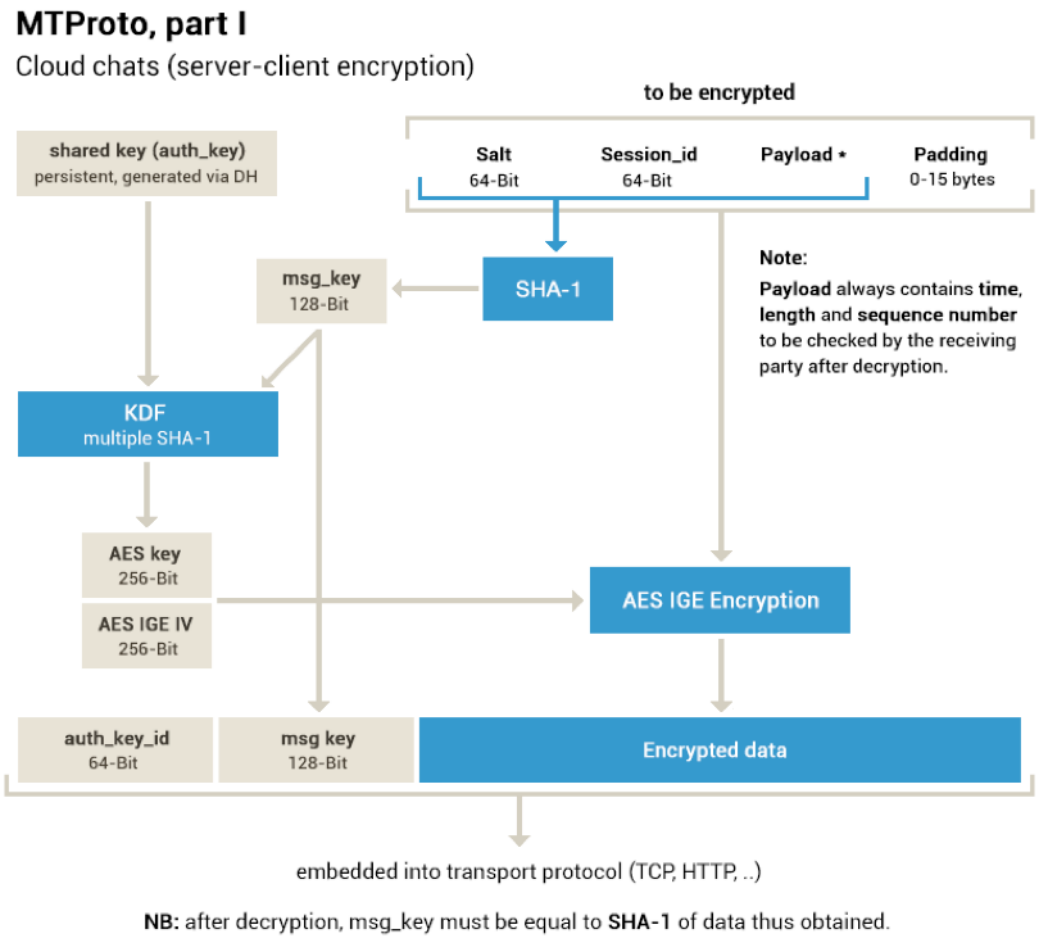

Figura C.1: Diagrama de funcionamiento del protocolo

Telegram obtiene su huella digital mediante el algoritmo SHA-1. El resultado es una clave que identifica al mensaje (msg_key). Pero además ese resultado se combina junto con la clave compartida entre cliente y servidor. Esta clave compartida es intercambiada en el primer registro del cliente mediante el algoritmo de intercambio Diffie-Hellman, que permite que ambas partes lleguen a una clave común secreta, sin tener que transmitirla. Así, se obtiene una nueva clave con la que cifrar el mensaje. Se ve en este momento que la clave con la que se cifra el mensaje depende del contenido del mensaje (algoritmo robusto).

Tras cifrar el mensaje, se envía un paquete que contiene el mensaje cifrado (Encrypted data), la huella digital SHA-1 del texto sin cifrar (msg_key) y un número que identifica la clave compartida que se usa (auth_key_id).

Al recibir el mensaje en el otro lado de la conexión, se recrea la clave de cifrado usando la clave compartida y la huella del mensaje, se descifra el texto y se comprueba que la huella digital concuerda. También se comprueba que el salt sea correcto (igual al que el servidor haya definido), que la fecha y hora sea razonable y que el número de secuencia sea el apropiado.

Así, con esa comprobación múltiple, Telegram se asegura que nadie más puede leer el mensaje y que además ha llegado sin ningún tipo de modificación [3][4].

Anexo D

OAuth2

OAuth2 es un protocolo de autorización que permite a terceros (clientes) acceder a contenidos propiedad de un usuario (alojados en aplicaciones de confianza, servidor de recursos). Es decir, que aplicaciones de terceros pueden acceder a contenidos propiedad del usuario, sin estas aplicaciones conozcan las credenciales de autenticación. Es el protocolo de funcionamiento que se ha utilizado para acceder a los datos alojados en la cámara. En nuestro escenario OAuth2 hay tres partes claramente identificadas:

+ **Usuario final:** Entidad que se comunica con el cliente para lograr el acceso a los recursos protegidos.
+ **Cliente:** La aplicación bot integrada en Telegram es la que hace peticiones a recursos protegidos en nombre del propietario de recursos (Usuario) con la autorización del mismo.
+ **Servidor de recursos:** Es la entidad que tiene los recursos protegidos. Es capaz de aceptar y responder peticiones (video, información, fotografías, etc) usando el access token que debe venir en el cuerpo de la petición.
+ **Servidor de autorización:** El servidor de autenticación es el responsable de generar tokens de acceso y validar usuarios y credenciales.

Una vez definido el escenario, la aplicación implementa tres funcionalidades a través de este protocolo:

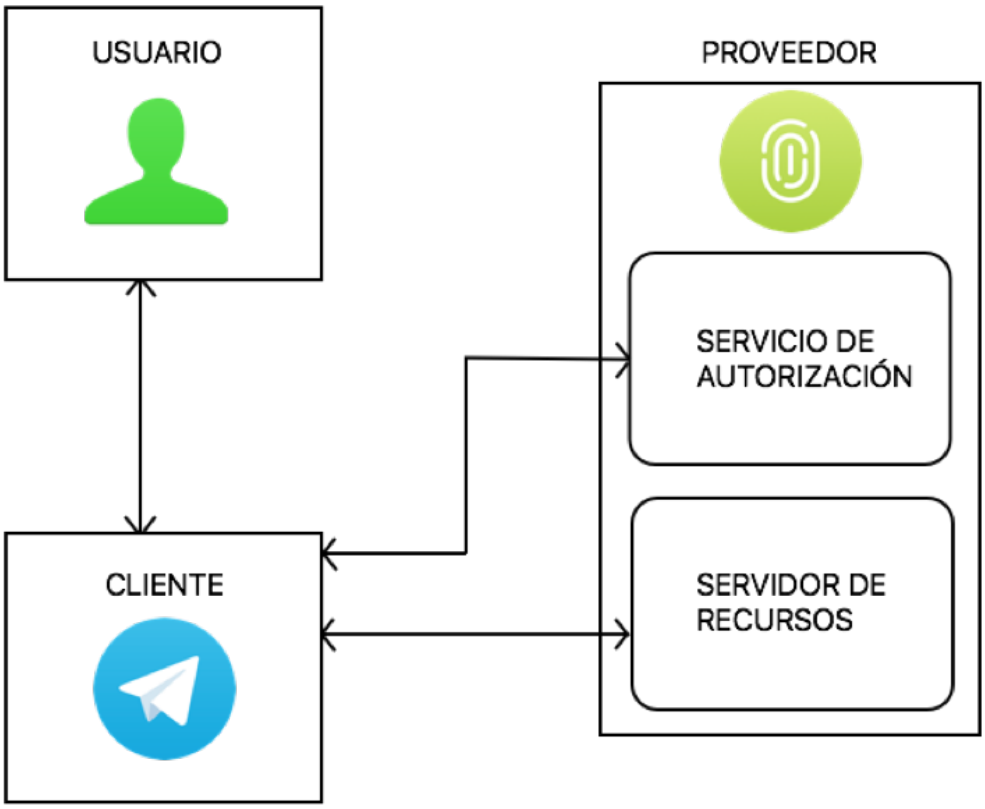

Figura D.1: Arquitectura Autorización Aplicación

1. El cliente solicita autorización al propietario del recurso (en nuestro caso no es necesario ya que el cliente es el propio usuario). **2.** El propietario del recurso devuelve un método de autentificación válido (en nuestro caso utilizaremos las credenciales de netatmo como método de autentificación)
3. El cliente pide al servidor de autorización un token de acceso, presentando las credenciales del paso 2.
4. El servidor de autorización devuelve un token de acceso válido.
5. El cliente y el servidor de recursos ya son capaces de intercambiar peticiones seguras con el token de acceso para servir contenido protegido.

sectionObtención del token Los token de acceso deben tener un periodo de expiración después del cual se consideran caducados y el proveedor debe rechazarlos, obligando al cliente a obtener un nuevo token de acceso. Para

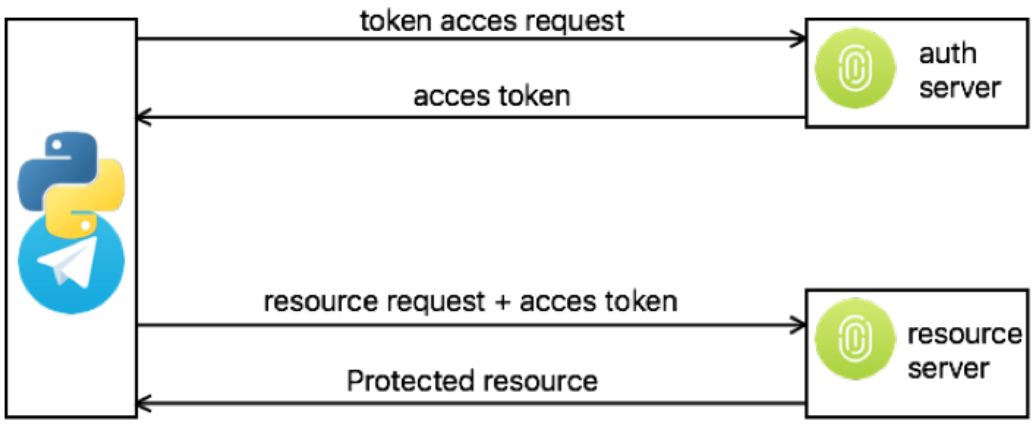

Figura D.2: Concesión del token

evitar tener que volver realizar todo el proceso de validación necesario para la obtención del token, entra el mecanismo de Refresh Token. Cuando el proveedor concede un token de acceso, puede incluir un token de refresco asociado al token de acceso. Mediante este token de refresco se puede obtener un nuevo token de acceso válido siguiendo el siguiente flujo:

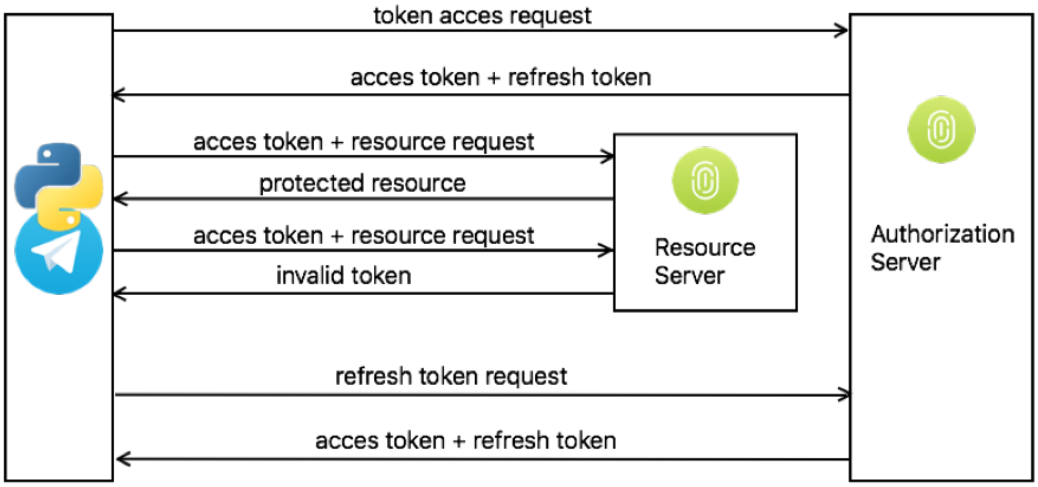

Figura D.3: Proceso de refresco del token

1. El cliente pide un token de acceso al servidor de autorización

2. El servidor valida la petición y envía los token de acceso

3. El cliente pide los recursos al servidor de recursos identificando con el token de acceso, si es válido, el servidor devuelve el contenido

4. Pasado un tiempo el servidor detecta el token de acceso con el timestamp expirado o inválido. El servidor se lo indica al cliente

5. El cliente hace una petición para obtener un nuevo token de acceso presentado el token de refresco obtenido en el paso 1.

6. El servidor de autorización autentica al cliente validando el token de refresco. Devuelve un acces token válido.

7. Se reanuda la y comunicación con el servidor de recursos y se cursan las peticiones.

Anexo E

API Netatmo

Netatmo es una empresa que comercializa productos como termostatos inteligentes, estaciones meteorológicas y cámaras de vigilancia. La cámara en el proyecto utilizada cuenta con las siguientes especificaciones:
+ Sensor de video de 4MP con resolución de 1920x1080
+ Visión nocturna LED Infrarrojos.
+ Conexión Wi-Fi 802.11a/b/g/n,
+ Puerto Ethernet 10/100Mbits
+ Ranura microSD (máximo 32gb).
+ Dimensiones 45x45x155 mm

Dispone de una API oficial para el desarrollo de aplicaciones y está traducida parcialmente a Python. La API [5] funciona a través de ?scopes? que se seleccionan según el interés del usuario:
- '**read_camera**': permite acceder a eventos guardados y timelines.
- '**write_camera**': permite sets del estado de las personas grabadas en el sistema
- '**acces_camera**': permite acceder a streamings en vivo, realizar capturas, o descargar grabaciones almacenadas en la tarjeta microSD.
- Además, incluye información del usuario como las unidades métricas utilizadas, el idioma, y el huso horario. También almacena información de

estado del dispositivo.

Para acceder a los datos que enmascaran dichos '**scopes**' es imprescindible acceder con el parámetro token de acceso. En el proyecto se hace uso de un protocolo de autenticación denominado OAuth2, (su funcionamiento está detallado en el Anexo D) que se encarga de expedir y renovar los token.

La API funciona a través de eventos, cada detección de sus sensores se considera un evento, que está clasificado en una tipología concreta (persona, movimiento, etc.). Cuando el programa detecta movimiento, crea un nuevo evento y acciona una señal que activa la grabación, este podrá ser almacenado localmente en una tarjeta microSD cifrada o también añadiendo redundancia en servicios de alojamiento como Dropbox o servidores FTP. La gestión del almacenamiento de la tarjeta microSD seguirá un modelo de colas FIFO donde los vídeos más antiguos se eliminarán periódicamente. Para cada evento se generan metadatos que identifican todas sus propiedades y se almacenan en estructuras JSON.

Anexo F

Modelos de Información

Como se ha visto en el Capítulo 3, el sistema utiliza cuatro registros JSON para mantener la información ordenada de los usuarios activos en el sistema. En detalle se organizan:

1. Registro listener que almacena la interacción de cualquier usuario:

```
+--ro history
|   +--ro año                 string  -> 2017
|   +--ro día                 string  -> 15
|   +--ro mes                 string  -> 10
|   +--ro hora                string  -> 20:21:12
|   +--ro nombre de usuario   string  -> Arturo
|   +--ro identificador       int     -> 1679055044
|   +--ro mensaje             string  -> Muéstrame el video en directo
```

2. Registro cameras que almacena todas las cámaras registradas en el bot y los usuarios que tienen acceso:

```
+--ro cameras
|   +--rw id camera           string  -> 70:ee:50:1d:54:8f
|   |   +--rw id user         int     -> 1679055044
```

3. Registro login que almacena las credenciales y la configuración personal de cada usuario registrado.

```
+--ro login
|  +--ro identificador          string  -> 1679055044
|  |  +--rw username            string  -> steve@gmail.com
|  |  +--rw password            string  -> contrasegna123
|  |  +--rw client_id           string  -> 8udsa898932dsaA...
|  |  +--rw client_secret       string  -> fdsjhZNkd9jskx...
|  |
|  |  +--rw acces_token         string  -> 5438292039490423...
|  |  +--rw refresh_token       string  -> 58ab6ed469f7409e20...
|  |  +--rw expiration          int     -> 12343233
|  |  +--rw numeroAvisos        int     -> 1
|  |  +--rw movimiento          bool    -> True
|  |  +--rw conocidos           bool    -> False
|  |  +--rw desconocidos        bool    -> True
|  |  +--rw avisos
|  |  |  +--nombre              string  -> Alex
|  |  |  +--hora inicio         int     -> 17
|  |  |  +--hora final          int     -> 20
|  |  |  +--visto               bool    -> True
```

4. Registro que almacena un histórico de todos los eventos de una cámara. Además del objeto JSON se almacenan los archivos jpg de todos los eventos identificados por el id de la cámara y el timestamp de dicho evento:

```
+--ro events
|  +--ro timestamp*             int     -> 1502355815
|  |  +--ro category            string  -> human, animal
|  |  +--ro video_status        string  -> available
|  |  +--ro video_id            string  -> 9b9b88f7-87e1-...
|  |  +--ro vignette
|  |  |  +--version             int     -> 1
|  |  |  +--id                  string  -> 598c2b6c2b2b46888c8b45ad
|  |  |  +--key                 string  -> 07d27b72e1930bf119e...
|  |  +--ro snapshot
|  |  |  +--version             int     -> 1
|  |  |  +--id                  string  -> 598c2b6c2b2b46888c8b45ad
|  |  |  +--key                 string  -> 07d27b72e1930bf119...
|  |  +--ro camera_id                   string  -> 70:ee:50:1d:54:8f
|  |  +--ro time                int     -> 1502355815
|  |  +--ro message             utf-8   -> Arturo visto
|  |  +--ro type                string  -> person, sd, home_away
|  |  +--ro sub_type            int     -> 1
|  |  +--ro id          string  -> 598e3190b26ddfe40c8b8f77
|
```

Anexo G

Telebot

La librería Telebot es uno de los elementos más importante para la gestión de nuestro programa y es el encargado de crear la relación entre nuestra aplicación y el servidor de Telegram. Incluye los métodos para la gestión de los mensajes recibidos, y las interacciones por parte del usuario con el cliente. Las peticiones al servidor de Telegram *https://api.telegram.org/bot¡token¿/METHOD_NAME* deben incluir el token del bot y el método al que llaman. Este método puede incluir parámetros, que deben ser informados en peticiones http GET o POST.

Para instanciar un objeto de la clase telebot necesitamos el mencionado token de acceso, este token es un string único que Telegram utiliza para identificar cada bot. Una vez la clase está instanciada pasa a su funcionamiento normal quedando encargada de las siguientes actividades:

- **send_message**: Elemento encargado de enviar al cliente los mensajes, es necesario incluir el identificador del usuario al que van dirigidos y el texto que contiene el mensaje. Si se quiere añadir formato al texto es posible señalar la codificación que utiliza. Permite añadir respuestas con teclados a través del parámetro reply_markup, que se utilizan en funciones como la creación de avisos o para listar usuarios (explicado más adelante).

- **edit_text_message:** Elemento encargado de editar los mensajes dirigidos al cliente. Se utiliza para mantener la cohesión en la línea temporal del programa. Se debe incluir el mensaje que modifica y el usuario al que se dirige.
- **send_photo**: El envío de archivos de imagen se hace a través de este elemento. Permite descargar imágenes al usuario.
- **send_chat_action:** Con el objetivo de mantener al usuario informado, está función permite notificar al usuario que el servidor está procesando información. El programa lo utiliza para señalar la conversión de archivos de video y la descarga de imágenes.
- **send_video:** Elemento que permite al usuario descargar los videos de los eventos grabados por la cámara. Cada petición de video sobrescribe la anterior con el fin de no degradar la capacidad de almacenamiento.
- **set_webhook:** Método utilizado para indicar al servidor de Telegram el URL al que debe notificar las actualizaciones. En la aplicación se establece la instancia virtual como receptor de estas peticiones.
- **set_update_listener**: La forma de gestionar las peticiones del cliente se gestiona con este método que llama a la función Listener cada vez que recibe una actualización.
- **process_new_updates**: Elemento que se encarga de recoger todas las actualizaciones que llegan a través del webhook y servirse a través del bot.

Anexo H

Reconocimiento Facial

En el hogar conectado, una cámara es un elemento central. Cuidando de la privacidad, puede resultar muy útil si incluye funciones como la grabación de video o el reconocimiento facial. El reconocimiento facial actual se basa en un proceso que consta de cuatro módulos principales que se apoyan en dos bases de datos, la primera contiene imágenes de múltiples caras con distintas poses, la segunda se conforma de la galería de aquellas caras que el sistema detecta [2.4]:

1. Face Detection: El algoritmo proporciona la localización y la escala de la cara en referencia a la fotografía.

2. Face Alignment: Se localizan las componentes de la cara y se normalizan mediante propiedades geométricas, que incluyen la normalización de la iluminación.

3. Face Description: se extraen detalles como la distancia entre las pupilas o la posición de la nariz.

4. Face Classification: el vector de características extraído se compara con los vectores de características extraídos de las caras de la base de datos. Si encuentra uno con un porcentaje elevado de similitud, nos devuelve la identidad de la cara; si no, nos indica que es una cara desconocida.

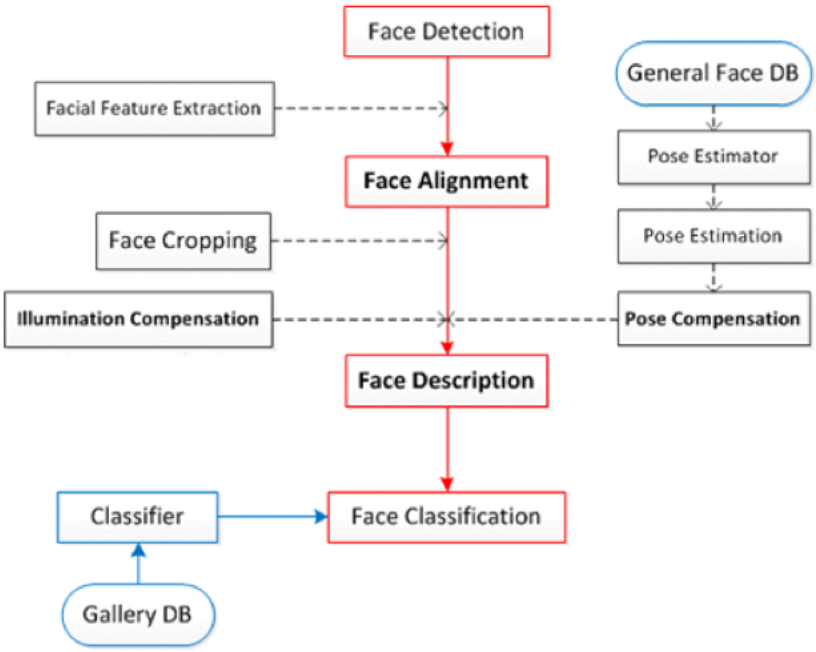

Figura H.1: Diagrama de flujo Reconocimiento facial.