Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

HomeSurveillance:
studio e realizzazione di un sistema di videosorveglianza a basso costo con riconoscimento delle sagome umane

Relatore:
Chiar.mo Prof.
Renzo Davoli

Presentata da:
Adamo Fapohunda

III Sessione
Anno Accademico 2016/2017

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Con il termine sorveglianza si indica l'attività del vigilare su un luogo o comunque un bene, spesso con l'ausilio di reti di sistemi di allarme, videocamere e sensori[47]. In sostanza si indica l'azione di acquisire informazioni sull'ambiente circostante per poter prendere decisioni appropriate.

Possiamo suddividere lo sviluppo dei sistemi di sorveglianza in base al quantitativo di informazioni ottenute e al numero di operatori necessari per elaborarle e distribuirle[57].

Fino a prima della comparse del telegrafo i sistemi di sorveglianza prevedevano l'impiego di addetti per poter rilevare la presenza di eventuali minacce. L'acquisizione delle informazioni veniva effettuata dagli occhi e dalle orecchie di chi era preposto a tal compito (vedette e sentinelle), così come la distribuzione delle stesse.

Tuttavia l'invenzione di strumenti per la trasmissione di dati a grandi distanze rese possibile inviare prima semplici segnali e poi audio e video. Ciò nonostante la sorveglianza era ancora un'attività legata all'elemento umano. Di fatto era ancora necessario un buon numero di operatori addetti all'analisi dei video[57].

Con l'abbassarsi del costo di sensori è stato possibile integrarli nei sistemi di videosorveglianza, fornendo un maggior numero di informazioni e permettendo di controllare vaste aree con un numero sempre minore di personale.

Ma la svolta più importane si ebbe con lo sviluppo dell'informatica. Di fatto, in poco più di un cinquantennio, si è passati dal teorizzare dei test

sull'intelligenza delle macchine al realizzare algoritmi in grado di battere un giocatore professionista ad un complesso gioco da tavolo[32]. L'ausilio dell'intelligenza artificiale ha reso i sistemi di sorveglianza quasi del tutto indipendenti dall'intervento umano.

In aggiunta, mentre negli anni settanta Internet era appannaggio di accademici e militari, a partire dagli anni 2000, questa entrò nelle case dei comuni cittadini[22]. L'utilizzo di Internet ha permesso la completa integrazione dei sistemi di video sorveglianza con le reti di calcolatori. Si assiste, nel corso degli anni, alla diffusione di sistemi di sorveglianza a basso costo diretti al mercato dei privati cittadini[47].

Tuttavia la maggior parte delle soluzioni attualmente in commercio, oltre a non beneficiare dell'utilizzo dell'intelligenza artificiale, salve le informazioni degli utenti in sistemi di archiviazione di massa sui quali l'utente non ha controllo né proprietà (Capitolo 1).

Scopo di questo progetto di tesi è implementare un sistema di sorveglianza domestico intelligente alternativo alle soluzioni proprietarie. Si vuole fornire una soluzione ibrida che sfrutti sensori e intelligenza artificiale per avvisare istantaneamente l'utente una volta rilevata la minaccia.

Nella prima parte verranno passate in rassegna le principali soluzioni per i sistemi di sorveglianza con relativa critica. A seguire verrà illustrata la soluzione realizzata spiegandone struttura, principali tecnologie utilizzate e le motivazioni che hanno portato all'adozione di queste. Successivamente la messa in prova del prototipo e l'analisi delle criticità e dei consumi. Infine le conclusioni.

Capitolo 1

Stato dell'arte

La maggior parte dei sistemi di sorveglianza sono sistemi analogici di telecamere a circuito chiuso *Closed-Circuit Television* (CCTV). Questi offrono ottime performance nella maggior parte delle situazioni e l'affidabilità di una tecnologia matura, ma richiedono la registrazione e la trasmissione dei dati in analogico, con tutti i limiti del caso[57]. In risposta a tale problematica, sempre più aziende stanno proponendo sistemi basati su telecamere connesse ad Internet (IPCamera) in grado di fornire lo *streaming* video in formato digitale e di essere controllate dalla rete stessa[5]. Il vantaggio di sistemi siffatti è la possibilità di accedere al segnale video potenzialmente da qualunque dispositivo in grado di gestire protocolli di rete. Di conseguenza, diventa possibile adoperare server di archiviazione di dati per il salvataggio delle riprese e fornire servizi in modalità *cloud*, ossia accessibili su richiesta attraverso una connessione ad Internet.

Lo storico marchio Foscam propone nella fascia bassa del catalogo la IP-Camera C2 [19]. Questa è dotata di sensore ad infrarossi passivo (PIR) per la rilevazione del movimento e di un led ad infrarossi per la registrazione notturna, inoltre offre la possibilità di salvare i video su una scheda di memoria SD in locale. La camera possiede un'interfaccia grafica accessibile da browser Web tramite la quale è possibile visualizzare lo *streaming*. Inoltre è disponibile un'applicazione per dispositivi mobili che permette di ricevere le

notifiche. Foscam offre la possibilità di salvare i dati sui server dell'azienda (in cloud) dietro pagamento di un abbonamento separato per ogni camera. Dunque avere più camere, implica il dover pagare più abbonamenti distinti.

Anche la celebre azienda Netgear ha messo in commercio la sua soluzione per la sorveglianza domestica: ARLO[39]. Tale sistema è completamente *wireless*, non solo per quanto riguarda la comunicazione ma anche per l'alimentazione: ogni camera è alimentata da quattro pile CR123 al litio. Anche queste possiedono un led ad infrarossi per illuminare le riprese notturne e un sensore PIR per rilevare il movimento. L'azienda offre gratuitamente (fino a cinque camere) la possibilità di salvare i video in cloud fino ad un massimo di sette giorni e la possibilità di ricevere notifiche push grazie all'applicazione per smartphone e tablet.

A distinguersi per l'approccio al problema è l'azienda EZVIZ[18]. Questa offre un sistema modulare che comprende una centralina WiFi a cui si possono collegare fino a 32 sensori. Le IPCamere sono collegate via WiFi, mentre i sensori aggiuntivi sono connessi tramite radiofrequenze. Ad ogni sensore è possibile collegare una camera. Anche in questo caso, le camere disponibili sono corredate di sensore PIR e led ad infrarossi. Le immagini catturate vengono salvate solo all'interno delle camere (tramite scheda di memoria SD) oppure su un eventuale dispositivo di archiviazione collegato alla rete (Network Attached Storage - NAS). Per quanto riguarda l'accesso da remoto, l'applicazione proprietaria permette di visualizzare lo *streaming* video in diretta e di ricevere notifiche.

Tutte le soluzioni proprietarie presentate hanno il grande difetto di dipendere fortemente dall'hardware e dal software del produttore. Non è possibile modificare le camere, né modificare il software in base alle proprie esigenze e in caso di cessato sviluppo, poi, il sistema rimarrebbe senza gli aggiornamenti di sicurezza. Inoltre, dipendere dal produttore, significa doversi anche fidare della riservatezza e della sicurezza di questo il che potrebbe rappresentare una potenziale minaccia al diritto che un utente ha di disporre delle proprie informazioni[17].

Fortunatamente esiste un'alternativa al mondo proprietario: l'open hardware-software. Secondo la GNU Software Foundation, un programma è software libero se gli utenti godono delle quattro libertà fondamentali[20]:

- Libertà di eseguire il programma come si desidera, per qualsiasi scopo (libertà 0).
- Libertà di studiare come funziona il programma e di modificarlo in modo da adattarlo alle proprie necessità (libertà 1). L'accesso al codice sorgente ne è un prerequisito.
- Libertà di ridistribuire copie in modo da aiutare il prossimo (libertà 2).
- Libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti da voi apportati (e le vostre versioni modificate in genere), in modo tale che tutta la comunità ne tragga beneficio (libertà 3). L'accesso al codice sorgente ne è un prerequisito.

Sulla scia dell'open software, è nato il concetto di open hardware. La possibilità di avere software open source a costo irrisorio quando non nullo, ha permesso agli amatori di poter progettare i propri circuiti in casa e grazie all'abbassamento dei costi della tecnologia è diventato possibile inviare via mail il proprio progetto ad un'azienda per stampare circuiti personalizzati a costi relativamente contenuti [1]. Progetti come Arduino e Raspberry Pi hanno reso l'accesso a microcontrollori e single board ancora più semplice e alla portata anche degli utenti inesperti.

Ma il mondo open non è rivolto ai soli amatori, sempre più aziende hanno abbracciato questa filosofia[15] come nuovo modello di business. La prima a farlo è stata Netscape, che nel 1998, spiazzò i suoi azionisti rendendo disponibile il sorgente del proprio browser web. La mossa fu senza precedenti, Netscape di fatto era un'azienda quotata in borsa, nulla a che vedere con gli sviluppatori dei poli di ricerca universitari. Tutti si chiesero come fosse possibile che un'azienda che regalavi software possa generare utili[24]. Da allora l'open source è passato dall'essere etichettato come "cancro" dal CEO

della Microsoft[3] ad essere fonte di valore per le aziende. L'open non è solo un modo per risparmiare sui costi delle licenze[4]: poter modificare il software permette alle aziende di muoversi in nuovi modelli di business come la consulenza, lo sottoscrizione e l'hosting. Inoltre, la risorsa più importante nel campo informatico è senza dubbio la conoscenza delle migliori metodologie per risolvere i problemi. Portare in azienda l'open software, significa portare un bagaglio di conoscenze e abilità operative[9].

Dunque accanto alle soluzioni proprietarie, sono nati diversi progetti che propongono sistemi di sorveglianza domestica più o meno open.

Alcuni utilizzano dei Raspberry Pi con un modulo camera e sensore PIR[2]. La notifica dell'utente avviene tramite email, il che implica che l'intrusione potrebbe essere segnalata troppo tardi.

In altri casi si utilizzano setup similari senza però l'ausilio di sensori PIR. Questo è possibile grazie all'utilizzo di algoritmi di *motion detection*. Tuttavia non rappresenta una soluzione ideale per via del consumo di CPU necessario per eseguire tale operazione [40].

Quelle presentate rappresentano la soluzione più semplice, ma per contro, non permettono in alcun modo di distinguere un'intrusione da un falso positivo (vedi Sezione 2.3.7).

I sistemi più evoluti utilizzano algoritmi di *object detection* per riconoscere eventuali intrusi. Sfortunatamente tali sistemi vengono spesso associati ad hardware propiertario. Vengono utilizzate telecamere USB montate sul Raspberry Pi [33], oppure vengno uilizzati server dedicati con l'algoritmo costantemente in funzione[28].

Scopo di questo progetto di tesi è studiare un sistema di sorveglianza domestico che sfrutti i sensori PIR per la fase di motion detection, un algoritmo di riconoscimento per accertarsi che si tratti di una minaccia e infine un servizio di messaggistica istantanea per avvisare l'utente.

Capitolo 2

Il Progetto HomeSurveillance

2.1 Idea

L'idea per il progetto nasce in risposta ad un'esigenza reale: poter avere un sistema di sorveglianza efficace in contesti che ospitano animali domestici.

Si vuole progettare un sistema in grado di soddisfare i seguenti requisiti funzionali:

- Rilevare eventuali movimenti.
- Accertarsi che sia un intruso.
- Avvisare l'utente.
- Fornire lo streaming video in tempo reale.

Su ispirazione dei modelli presenti in commercio, per assolvere alla prima funzione si è deciso di utilizzare un sensore PIR.

Per la seconda funzione,occorre un modo per esaminare lo *streaming* video accertandosi che ci sia effettivamente un intruso. Si è deciso di utilizzare OpenCV con una MobileNets 2.3.5. A partire dall'analisi dei progetti svolta nel Capitolo 1, è stata subito scartata la scelta di un servizio e-mail. Si è optato per l'utilizzo di un'applicazione di messaggistica istantanea: Telegram 2.3.6.

Per provvedere allo *streaming* video al di fuori della rete domestica, è necessario renderla accessibile e conoscerne l'indirizzo. Si è scelto di sfruttare lo Universal Plug and Play Support 2.3.8 e un Telegram Bot 2.3.6.

Un sistema di sorveglianza, generalmente, deve restare in funzione per lunghi periodi di tempo, motivo per cui si richiede che questo abbia dei consumi energetici più bassi possibile. Inoltre si vuole poter gestire più videocamere all'interno della stessa abitazione, senza dover gestire più sistemi di sorveglianza.

2.2 Panoramica del sistema

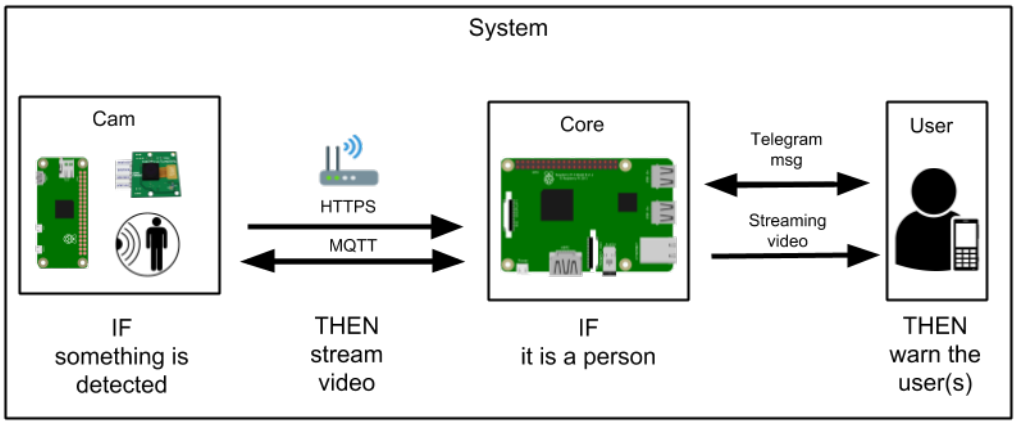


Figura 2.1: System Overview.

Sulla base dei requisiti presentati nella sezione precedente, è stato studiato un sistema composto da due tipologie di dispositivo: Cam e Core (Figura 2.1).

La Cam è la componente che funge da IPCamera ed è dotata di sensore PIR, mentre il Core è la componente che si occupa di gestire l'interazione con l'utente e di elaborare lo streaming video.

Il funzionamento è abbastanza semplice: la Cam grazie al sensore PIR rileva i movimenti, ma non è in grado di accertarsi se siano eventuali intrusi o falsi positivi. A questo punto invia un segnale al Core. Questo richiede

lo *streaming* del video e lo analizza. Se viene rilevata una sagoma umana, allora avverte l'utente.

Una volta stabilita la struttura logica del sistema, resta da definire quella fisica. Per quanto riguarda il Core si è optato per un Raspberry Pi 3 Model B. Questo permette di avere la potenza di calcolo necessaria all'analisi dei video ed inoltre funge da server per il Telegram Bot, per la comunicazione in MQTT e per il servizio di *streaming* al di fuori della rete domestica. Per quanto riguarda il dispositivo Cam, inizialmente si era optato per l'utilizzo di webcam USB. L'ipotesi è stata subito scartata facendo alcune considerazioni sui limiti fisici dei cavi USB. Questi sono progettati per portare il segnale fino a 5 metri. Anche utilizzando degli hub come ripetitori non si possono superare i 25/30 metri senza incorrere in grossi problemi di *timing* [43]. La soluzione al problema è l'utilizzo di una IPCamera. Si potrebbe pensare di acquistarne una, ma questo non permetterebbe di configurare eventuali servizi aggiuntivi come la sicurezza e le strategie di comunicazione e segnalazione di minacce. Si è dunque deciso di assemblare la Cam utilizzando un Raspberry Pi Zero W accoppiato ad un modulo camera ed un sensore PIR. Tale configurazione, offre l'indubbio vantaggio di poter fornire l'alimentazione tramite pacco batterie rendendo il dispositivo Cam del tutto *wireless* e quindi facilmente posizionabile in punti critici dell'abitazione. Inoltre attualmente tale soluzione ha costi inferiori alla maggior parte delle soluzioni in commercio.

Verranno ora passate in rassegna tutte le principali tecnologie sia hardware che software di cui il progetto si avvale. Infine ne verrà illustrato il funzionamento.

2.3 Principali tecnologie utilizzate

2.3.1 Raspberry Pi

Il Raspberry Pi è un singleboard computer, ossia un computer implementato su una sola scheda con processore, scheda grafica e ram. Il progetto

vede la luce nel 2012 in Inghilterra ad opera della Raspberry Pi Foundation. Dal 2012 ad oggi sono stati prodotti diversi modelli, per questo progetto si è deciso di utilizzare una versione 3 model B e una versione Zero W.

Raspberry Pi 3 Model B

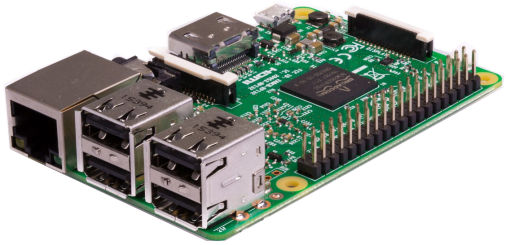


Figura 2.2: Raspberry Pi 3 Model B.

Il Raspberry Pi 3 Model B [21] rimpiazza il suo predecessore Pi 2 Model B con cui condivide buona parte dell'architettura ad eccezione del System-on-chip (Soc) Broadcom che passa dal modello Broadcom BCM2836 (con un cluster di ARM Cortex-A7 a 32-bit) al Broadcom BCM2837 (con ARM Cortex A53 a 64-bit). Il tutto è corredato da:

- 1 Gb di memoria RAM DDR2 condivisa con la scheda grafica (una Broadcom VideoCore IV)
- 4 porte USB 2.0
- Single-chip BCM43438 che offre connessione LAN wireless e Bluetooth Low Energy
- Uscita HDMI a dimensione standard
- Porta Micro USB per la ricarica
- Porta Ethernet 10/100 Mbit/s (RJ-45)

Raspberry Pi Zero W

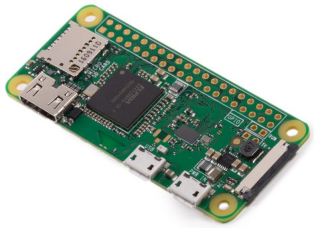


Figura 2.3: Raspberry Pi Zero W.

A differenza del fratello maggiore il Raspberry Pi Zero W monta lo stesso Soc single core (BCM2835) della prima versione del Raspberry Pi corredato di:

- 512 Mb DDR2 sempre condivisi con la scheda video (stesso modello del Pi3).
- Single-chip BCM43438 (stesso del Pi3)
- Porta la Micro HDMI per l'output video.
- Porta Micro USB per la ricarica
- Porta Micro USB per la connessione di periferiche

Caratteristiche Comuni

Entrambi i modelli inoltre offrono:

- Porta CSI per connettere una camera con bus dedicato (a differenza delle porte usb)
- Porta DSI per connettere un display touchscreen

- Porta Micro SD per installare il sistema operativo

Ma la caratteristica forse più importante dei Raspberry Pi è la possibilità di avere un'interfaccia di basso livello con ben 40 pin in configurazione 2x20 per General Purpose Input/Output (anche noto come GPIO), Serial Peripheral Interface (SPI), integrated circuit (IC), Universal Asynchronous Receiver-Transmitter (UART) e alimentazione a 3,3 e 5 Volt.

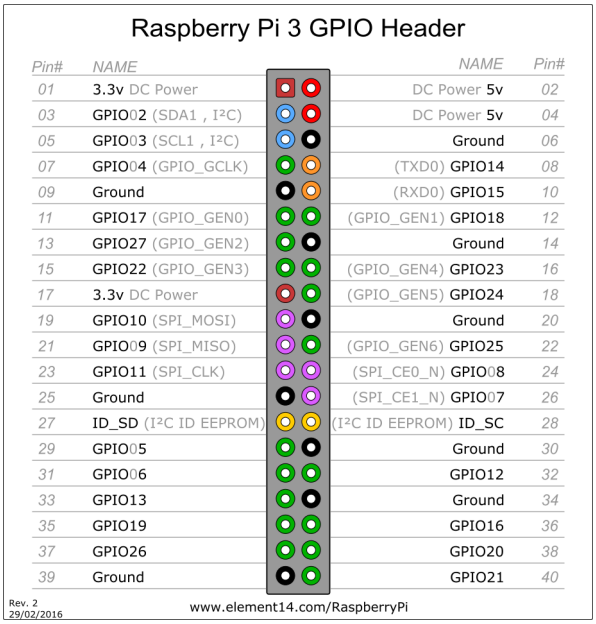


Figura 2.4: GPIO.

Inoltre esistono svariate distribuzioni linux compilate per essere eseguite su macchine ARM. In particolare verrà utilizzata una derivata di Debian ottimizzata per i Raspberry Pi chiamata Raspbian.

2.3.2 Modulo Camera

In prima istanza è stata valutata la possibilità di utilizzare una webcam USB, ma l'ipotesi è stata scartata per due ragioni: i consumi e l'impossibilità di avere un BUS (Binary Unit System) dedicato. Si è dunque optato per l'utilizzo della porta Camera Serial Interface (CSI). Nonostante fosse disponibile una camera assemblata e distribuita dalla Raspberry Pi Foundation, si è

scelto di utilizzare una camera Kuman. Questa offre le stesse caratteristiche della camera originale, ossia sensore da 5MP possibilità di riprendere video fino a 1080p a 30 fps, ma presenta la possibilità di avere una lente a fuoco regolabile, è priva di filtro per i raggi infrarossi (IR) ed è corredata da due led IR con fotocellula per le riprese notturne[30].

Figura 2.5: Kuman 5MP 1080p HD Camera Module.

2.3.3 MJPG-Streamer

E' un programma implementato in C che, come recita la documentazione, è in grado di copiare frame JPEG da uno o più input plugin a uno o più output plugin[34] . In particolare possiede un plugin studiato ed ottimizzato per prendere l'input dal modulo camera del Raspberry Pi e uno per restituire l'output sfruttando il protocollo HTTP. Può dunque essere usato per fare *streaming* di frame JPEG verso qualunque destinatario in grado di supportare stream MJPG, compresi i browser. Inizialmente il progetto venne ideato per funzionare su dispositivi *embedded*, pertanto è pensato per ridurre al minimo l'utilizzo di CPU e RAM. Sfortunatamente il programma non implementa nessun tipo di sicurezza di default, quindi qualunque altro utente collegato alla stessa rete sarò in grado di accedere alle immagini.

Per aggiare il problema si è scelto di utilizzare la tecnica di tunneling offerta da stunnel[50]. Una descrizione dettagliata verrà fornita nella Sezione 3.4.

2.3.4 MQTT

Dato che il tempo di invio/ricezione dei messaggi è alla base dell'efficienza e dell'affidabilità del sistema, si vuole poterlo fare nel modo più rapido possibile e con costo, sia in termini di ampiezza di banda (bandwidth) che computazionali, più basso possibile.

I principali paradigmi di comunicazione utilizzati a tal scopo sono di tipo request/response (polling) e publish/subscribe (push-based). HTTP, ad esempio, funziona con il modello request/response. Per lo scambio di messaggi si è deciso di utilizzare il modello publish-subscribe che permette la comunicazione asincrona fra diversi processi, oggetti o altri agenti. In sostanza il mittente (publisher) si registra come tale presso un terzo componente detto broker specificando l'argomento sul quale pubblica messaggi (topic). Il destinatario (subscriber), in egual modo, si registra dichiarando di essere interessato ad un determinato topic. Sarà il broker a filtrare i messaggi in base al topic e a consegnarli a tutti i diretti interessati.

Così facendo si ottiene il disaccoppiamento tra publisher e subscriber: non devono essere a conoscenza l'uno dell'altro, possono non essere connessi nello stesso momento, le operazioni non sono bloccanti (asincrone). Si potrebbe usare HTTP per costruire un modello publish-subscribe, tuttavia esiste un protocollo progettato per essere più leggero di HTTP 1.1 : MQ Telemetry Transport (MQTT)[26].

MQTT nasce nel 1999 ad opera di Andy Stanford-Clark di IBM, e Arlen Nipper di Cirrus Link Solutions. E' un protocollo Machine-to-Machine (M2M) con architettura publish-subscribe, posizionato sullo stack al Trasmission Control Protocol / Internet Protocol (TCP/IP)[37], come mostrato in Figura 2.6.

Esso permette di inviare e ricevere dati (payload) con un overhead basso, inoltre implementa un meccanismo di filtraggio dei messaggi in base al topic. Nella documentazione di MQTT 3.1.1 il broker è anche chiamato server[38]. Il funzionamento è quello spiegato in precedenza: publisher e subscriber sono client MQTT che hanno come unica responsabilità quella di doversi connet-

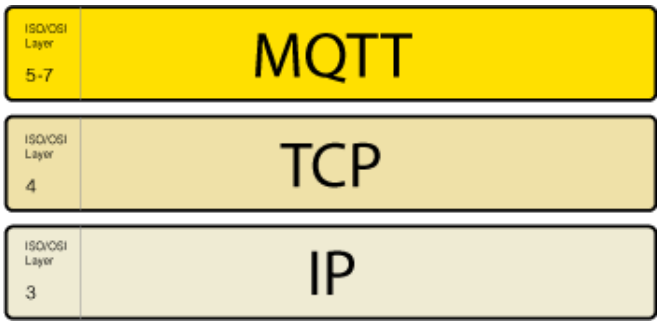


Figura 2.6: MQTT Stack.

tere al server specificando il topic. Un client può essere sia publisher che subscriber allo stesso tempo. Tutti i dispositivi in grado di gestire lo stack TCP/IP possono essere client MQTT.

Per rendere più chiaro il meccanismo, consideriamo un esempio in cui abbiamo un tre client: un Raspberry Pi con un sensore di temperatura, uno smartphone ed un tablet. Il tablet e lo smartphone si sottoscrivono al topic "sensor1/temperature" (come mostrato in Figura 2.7).

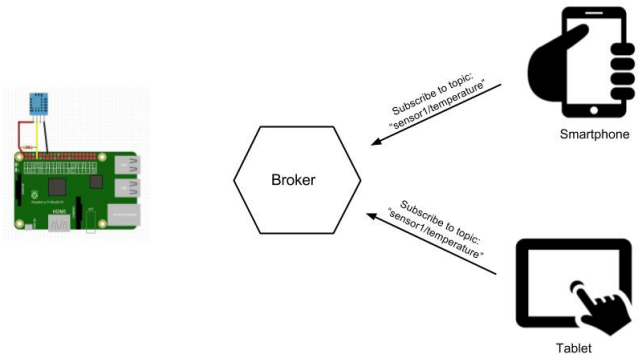


Figura 2.7: Registrazione di smartphone e tablet come subscriber.

Successivamente il Raspberry Pi legge il valore della temperatura, in questo caso 30 gradi (payload), e lo pubblica sul topic "sensor1/temperature".

Il broker si occuperà di inoltrare il payload verso tutti i subscriber di quel topic (Figura 2.8).

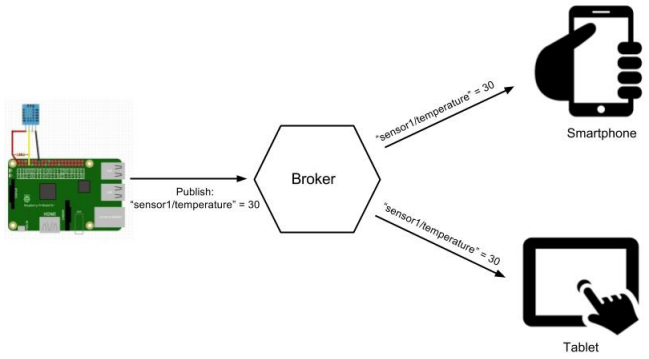


Figura 2.8: Registrazione del Raspberry Pi come publisher.

Le connessioni avvengono sempre da un cliente, sia publisher che subscriber, verso il server/broker. Per stabilire una connessione, il client manda un messaggio detto CONNECT al broker e questo, fatti i dovuti controlli, risponderà con un CONNACK. La connessione rimarrà aperta fino a quando non viene persa o il client non richiede la disconnessione inviando un messaggio DISCONNECT. E' anche possibile specificare un flag (ClientSession) per impostare il comportamento del server in caso di disconnessione. La sottoscrizione come publisher o subscriber avviene in modo analogo tramite l'invio dei rispettivi messaggi PUBLISH e SUBSCRIBE con relativo messaggio di acknowledgement. A differenza del pacchetto CONNECT con questi ultimi è possibile gestire la qualità del servizio (QoS):

- 0, At most once delivery: viene mandato il messaggio senza ricevere nessuna garanzia di ricezione sfruttando il più basso overhead possibile.

- 1, At least once delivery: il messaggio viene inviato e viene notificata l'avvenuta ricezione. Può generare duplicati dei messaggi.

- 2, Exactly once delivery: viene garantito che il messaggio venga inviato una sola volta al destinatario. Questo settaggio ha l'overhead più alto e la richiesta maggiore di banda in quanto richiede un doppio scambio di messaggi tra mittente e destinatario

Esistono svariate librerie client per MQTT. Il progetto utilizza Phao MQTT[7] sviluppato da Eclipse e open-source. Per quanto riguarda il server si è scelto di utilizzare sempre una libreria sviluppata da Eclipse: Mosquitto[6].

2.3.5 Human Detection

Per il riconoscimento della sagoma umana (Human Detection) viene utilizzata una tecnica di Deep Learning (DL). Come mostrato in Figura 2.9 il

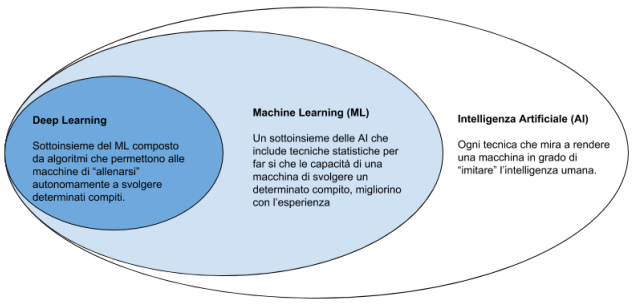


Figura 2.9: Diagramma esplicativo Intelligenza Artificiale, Machine Learning e Deep Learning

DL è un sottoinsieme del Machine Learning (ML) che a sua volta è un sottoinsieme dell'Intelligenza Artificiale (Artificial Intelligence, AI). Essenzialmente, AI è il termine generico con cui si definisce ogni tecnica che mira a riprodurre la facoltà di decisione dell'uomo coinvolgendo capacità come il ragionamento, l'apprendimento, la comunicazione e l'azione. Con Machine Learning si definisce una metodologia che fa uso della statistica per fornire una stima di

una funzione complessa in grado di migliorare tramite l'esperienza. Più precisamente "Un algoritmo apprende dall'esperienza E riguardanti una classe di problemi T con una misura pari a P, se la sua performance sui problemi T, misurata tramite P, aumenta con l'esperienza E[35]". Dunque il ML è un insieme di tecniche che consentono ad una macchina di imparare dai dati e di prendere decisioni su di essi. Con Deep Learning si intende la sottocategoria di ML che fa uso di Deep Neural Network (DNN). Le DNN forniscono un modello del cervello biologico con alcune differenze: il numero degli strati di neuroni è finito e la direzione di propagazione dell'informazione è prestabilita. L'affidabilità e la precisione delle decisioni prese dalle macchine è fortemente dipendente dalla quantità di dati (train-set) con i quali queste si "addestrano". Nel caso in cui si faccia uso del DL per il riconoscimento di oggetti/persone, le principali metodologie utilizzate sono:

- Faster R-CNNs[16]
- You Only Look Once (YOLO)[13]
- Single Shot Detectors (SSDs)[14]

Faster R-CNNs probabilmente è la più utilizzata e precisa, ma sfortunatamente non gode di buone performance. Per quanto riguarda YOLO, il problema è esattamente inverso, ottime performance ma bassa precisione. Le SSDs si posizionano a metà tra le due precedenti. Si è dunque optato per l'utilizzo di una SSDs sfruttando l'architettura MobileNets[10] sviluppata da Google e pensata per l'utilizzo su dispositivi mobili che per loro natura hanno risorse limitate. Il modello che è stato utilizzato è la versione Caffe di TensorFlow "addestrata" dall'utente di github chuanqui305[36]. Tuttavia non è possibile utilizzare il modello con qualunque immagine, queste devono essere prima normalizzate affinché la MobileNets possa prendere in input. Per questa operazione ci si avvale del modulo dnn di OpenCV.

2.3.6 Telegram Bot

Telegram è un servizio di messaggistica istantaneo basato sul paradigma di archiviazione cloud nato nel 2013 ad opera dei Fratelli Durov gli stessi creatori del social network russo VK[49]. Tra le caratteristiche principali, questo offre la possibilità di stabilire conversazioni cifrate end-to-end (ma solo nelle versioni del programma che salvano le conversazioni client-side), possibilità di inviare e ricevere qualunque tipo di file fino a 1,5 GB, di creare pacchetti di stickers personalizzati e di effettuare chiamate vocali cifrate[54].

Esistono centinaia di servizi simili; tuttavia quello che contraddistingue Telegram dalla concorrenza (oltre alle *features* di sicurezza) è la possibilità di creare dei Bot (abbreviazione di robot) utilizzando le API (Application programming interface) che l'azienda stessa rende disponibili. Le API così come i protocolli di comunicazione e il codice dei client sono pubblici e liberi. Questo implica che è possibile scrivere la propria applicazione client e farla interagire con quella di altri utenti sfruttando i servizi di Telegram. Un Telegram Bot è un *machine user*, ossia è un utente del sistema controllato da un macchina. Spesso vengono integrati con *features* di AI (Artificial Intelligence), ovvero vengono implementati meccanismi capaci di fornire alla macchina prestazioni indistinguibili dalle capacità umane. A differenza di un utente regolare, i Bot non necessitano un un numero di telefono per essere attivati, fungono semplicemente da interfaccia per il codice in esecuzione su una determinata macchina. Per creare un nuovo Bot, ancor prima di scrivere del codice, è necessario registrarlo. Per tale operazione ci si può affidare ad uno strumento offerto da Telegram stessa: BotFather. L'operazione è piuttosto semplice, basta accedere al client Telegram da qualunque dispositivo che lo supporti ed avviare una conversazione con il BotFather (Figura 2.10). Questo risponderà con un codice identificativo detto "token" che svolge lo stesso ruolo che numero di telefono svolge per gli utenti regolari: identificazione univoca del client nel sistema. Tra i vincoli a cui i Bot sono soggetti vi è l'impossibilità di iniziare una conversazione autonomamente, quindi dovrà sempre essere un utente regolare a inviare il primo messaggio. Per convenzio-

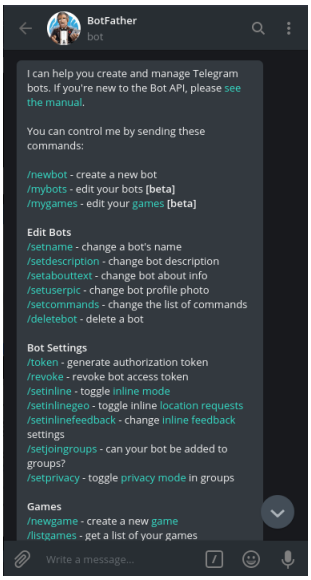


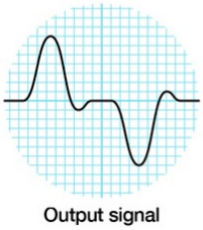
Figura 2.10: Esempio di BotFather in esecuzione su desktop client

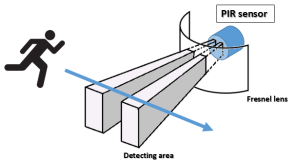
ne primo comando per iniziare una *chat* con un Bot è il comando "/start". Le API specifiche per i Bot sono un insieme di funzioni per supportare la creazione e la gestione dei Bot mediante diversi linguaggi di programmazione come Python, C/C++, Java e Lua[53] . Nell'ambito del progetto, si è deciso di utilizzare il linguaggio Python3 associato alla libreria python-telegram-bot in virtù della comunità che la supporta e della licenza LGPL-3[55]. Il Bot è stato utilizzato come interfaccia per il sistema di sorveglianza (attivazione/-disattivazione/richiesta di stream video) nonché da servizio Dynamic DNS (DDNS). Per spiegare l'utilità di tale servizio è necessario introdurre alcuni concetti. Internet è una rete composta da sotto reti con struttura gerarchizzata. Quando un utente intende connettersi ad Internet lo fa attraverso un Internet Service Provider (ISP), ossia ad un fornitore di servizi internet[31]. Questi rappresentano i nodi di accesso alla rete e sono organizzati gerarchicamente in base al tipo di copertura ad ai tipi di servizio che offrono. Ogni occupante di un livello gerarchico è detto essere fornitore degli utenti del livello sottostante e a sua volta utente dei livelli superiori. L'ultimo livello nella gerarchia è occupato dai router che offrono l'accesso alle reti locali

domestiche e aziendali (LAN). Se si vuole accedere ad un terminale di una rete LAN collegata ad un certo ISP, da un terminale collegato ad un'altra LAN, è necessario conoscere l'indirizzo IP del destinatario. Per questioni architetturali, gli indirizzi IP dei router domestici ricevono un indirizzo IP diverso ad ogni riconnessone. Per risolvere tale problematica esistono servizi detti Dynamic DNS, in grado di associare un nome di dominio (DNS) ad un determinato host per permettere il raggiungimento di questo attraverso il nome indipendentemente dall'indirizzo[52]. Per evitare di dover registrare un dominio, si è deciso di utilizzare Telegram per ottenere l'indirizzo attuale della macchina su cui è in esecuzione il Bot.

2.3.7 PIR Sensor

Come già spiegato, per rilevare il movimento, viene utilizzato un sensore ad infrarossi passivo (PIR sensor, acronimo di Passive InfraRed). Il funzionamento sfrutta l'effetto piroelettrico, ossia "l'effetto per cui si forma un temporaneo accumulo di cariche elettriche di segno opposto sulle facce opposte di certi cristalli in risposta ad un cambiamento di temperatura" [45]. Il sensore è composto da uno o più componenti piroelettrici, ogni componente è fatto di un materiale sensibile alle radiazioni infrarosse (IR). Quando il sensore non è attivo, i componenti rilevano lo stesso quantitativo di radiazione IR. Appena un corpo caldo attraversa il campo di vista del sensore, viene generata una differenza di potenziale positiva tra le due facce. Quando il corpo caldo esce fuori dal campo, viene generata una differenza di potenziale negativa. Questi due cambiamenti rappresentano una rilevazione. Dunque il sensore non rileva il movimento ma solo bruschi cambiamenti di temperatura. Dato che il corpo umano, così come quello di ogni altro essere vivente, ha una temperatura superiore allo zero assoluto, emette radiazioni nello spettro infrarosso pertanto è in grado di attivare il sensore.





Nel progetto viene utilizzato un sensore HC-SR501. La scelta del modello è stata determinata da due considerazioni: il costo irrisorio e la copertura (montata dal produttore) con lenti che focalizzano i raggi IR sulla componente piroelettrica[23].

Come mostrato in Figura 2.11, esso è composto da:

- I tre pin per la connessione.
- Un jumper per selezionare la modalità di attivazione: un singolo evento o eventi multipli.
- Un selettore della sensibilità regolabile per essere attivato fino alla distanza 7 metri.
- Un selettore per il delay regolabile da 3 secondi a 5 minuti.

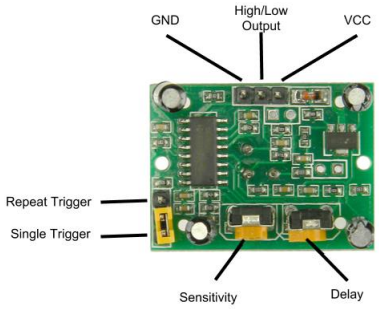


Figura 2.11: Sensore PIR HC-SR501.

2.3.8 Universal Plug and Play Support

Lo Universal Plug and Play (UPnP) è un insieme di protocolli di rete che permettono ai dispositivi collegati alla rete di rilevare reciprocamente la loro presenza e di avviare servizi[48]. Come è facile comprendere dal nome, l'obiettivo del protocollo è quello di semplificare il più possibile l'utilizzo delle reti domestiche. Quello che si vuole ottenere è la possibilità di visualizzare lo streaming video al di fuori della rete locale (LAN). Come già illustrato, per lo streaming video si è utilizzato il programma MJPG-Streamer che sfrutta il protocollo TCP (Transmission Control Protocol). Per usare TCP, la macchina che si appresta a inviare dati, apre un file speciale chiamato socket, che è identificato dall'IP della macchina stessa e da un numero di porta. Il ricevente si connette al servizio utilizzando il socket corrispondente[25].

Il problema nasce dal fatto che le macchine all'interno della LAN non possono essere raggiunte direttamente al di fuori della rete perché non possiedono un indirizzo IP pubblico. Tuttavia, come già illustrato, è possibile conoscere l'indirizzo del router che mette la LAN in comunicazione con Internet. Dunque occorre un metodo per poter mappare una porta di una macchina locale con una porta del router. Tale operazione è detta *port forwarding* o *port mapping*[46]. Un esempio di port forwarding è illustrata in Figura 2.12.

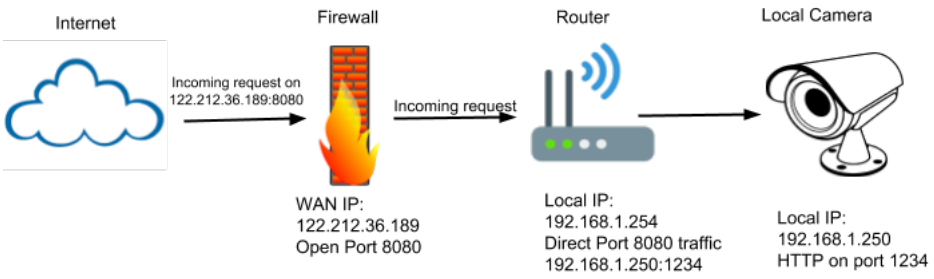


Figura 2.12: Esempio Port Forwarding

In questo caso il firewall è stato configurato per accettare connessioni sulla porta 8080 e il router è stato configurato per ridirigere il traffico in arrivo verso l'indirizzo 192.168.1.250 sulla porta 1234. Quando un utente fuori

dalla LAN vuole accedere alla IPCamera, invierà una richiesta all'indirizzo Wide Area Network (WAN) del router (122.212.36.189), assegnato dall'Internet Service Provider (ISP), specificando la porta di interesse (8080). A questo punto la richiesta passa dal firewall che controlla se la porta richiesta è stata "aperta" o meno. Se il firewall accetta la richiesta, sarà poi il router a inoltrarla all'indirizzo configurato, in questo caso la IPCamera. L'operazione di port forwardig può essere piuttosto macchinosa a seconda del router che viene utilizzato. Inoltre l'abilitazione del servizio espone la macchina locale ad eventuali attaccanti su Internet.

Si vuole poter aprire e chiudere le porte solo quando necessario. Per compiere questa operazione si sfrutta la libreria MiniUPnP installata sul dispositivo Core la quale fa uso del protocollo UPnP lato client.

2.4 Implementazione

Verrà ora esaminato più in dettaglio il funzionamento del sistema. I moduli principali che si occupano di tutte le funzioni sono il cameraSystem e il coreSystem.

Il primo va in esecuzione sui dispositivi Cam e istanzia le classi per gestire il client MQTT, il sotto processo che chiama MJPGStreamer e il sensore PIR. Il secondo va in esecuzione sul Core e istanzia il Bot Telegram e il client MQTT e la MobileNet per il riconoscimento delle immagini.

Le classi utilizzate dal cameraSystem sono:

- msgProcessor: si occupa di istanziare il client MQTT e di provvedere alla connessione con il broker.
- camera: si occupa di gestire il sottoprocesso MJPG-streamer
- motionSensor: si occupa di segnalare segnalare l'attivazione del sensore PIR

Le classi utilizzate dal coreSystem sono:

- detector: si occupa del processo di analisi del video.

- coreSys: si occupa di istanziare il Bot Telegram, il client MQTT e gestirne i messaggi.

Si è deciso di gestire il modulo cameraSystem mediante tre thread separati.

Il primo thread dopo aver inizializzato le classi necessarie, si metterà in attesa bloccante chiamando il metodo wait_for_motion offerto dalla classe MotionSensor [11]. Quando viene ricevuto il un segnale dal sensore PIR, l'istanza di motionSensor invia un messaggio di alert al Core (in realtà verrà inviato al broker che lo inoltrerà al Core). Per evitare che possa inviare ripetutamente messaggi di alert mentre il Core sta già analizzando il video, il thread viene nuovamente messo in attesa dell'evento che segnala la fine dell'operazione di streaming.

Il secondo thread si occupa dello streaming video. Come già accennato, questo fa uso del programma MJPGStreamer. Inizialmente il sistema era stato progettato per far partire il programma di streaming solo dopo aver ricevuto il messaggio di start dal Core, ma tale soluzione si è rivelata essere non ottimale in quanto il delay tra la ricezione del comando e l'avvio dello streaming superava i 5 secondi. Avendo preso atto della problematica, in prima istanza si è pensato di modificare il programma per poter gestire delle SIGPIPE. Tuttavia, analizzando il sorgente, si è scoperto che per evitare SIGPIPE generate dalla scrittura su socket, queste sono state disabilitate. Per risolvere il problema, si è pensato di far partire il processo di streaming all'avvio del sistema, aspettare che inizi a scrivere dati su una porta e poi bloccarlo con una SIGSTOP. Quando occorrerà accedere allo streaming, basterà inviare una SIGCONT. Questa metodologia, potrebbe lasciare nel buffer frame incompleti, ma ciò non costituisce un problema.

Il terzo thread si occupa della gestione dei messaggi in entrata/uscita del client MQTT e di sbloccare il tread di streaming e il thread che controlla il PIR. All'avvio, viene chiamato il metodo loop_forever() fornito dalla libreria

Paho[8]. Questo blocca il thread in attesa di ricevere messaggi e gestisce in automatico la riconnessione con il client.

Per quanto riguarda il modulo coreSystem, anche qui si è deciso di utilizzare tre thread. Il primo si occupa della segnalazione all'utente dell'eventuale interruzione degli altri thread.

Il secondo si occupa della gestione dei messaggi diretti al Bot Telegram. All'interno di un ciclo, viene chiamata la funzione bot.getUpdates della libreria telegram. A questa viene passato in input l'update_id e un valore numerico (timeout). Il primo valore è utilizzato per ricevere gli update dai server di Telegram usando la tecnica di long polling. In tal modo il server non chiude la richiesta quando non ha update da restituire. Il secondo valore, specifica per quanti secondi restare in attesa della risposta del server. Quando la richiesta non viene esaudita nel tempo richiesto, viene generata un'eccezione (telegram.error.NetworkError)[53].

Il terzo thread, in modo analogo al modulo cameraSystem, gestisce il client MQTT ed inoltre invia messaggi di notifica all'utente tramite i metodi dell'istanza del Bot.

Per semplicità si è deciso di suddividere la descrizione del sistema in due fasi: l'utente che richiede lo *streaming* del video e una Cam che rileva un movimento.

Nel primo caso, come illustrato in Figura 2.13, quando l'utente richiede lo streaming video tramite l'invio del comando /stream al Telegram Bot, questo in prima istanza si accerta che l'utente sia registrato (ne viene salvato l'id all'interno del file di configurazione). Se l'utente è abilitato il coreSystem mappa una porta del router con una porta di ogni dispositivo Cam. Conclusa l'operazione spedisce il comando di streaming e invia all'utente un link per ogni camera presente nel sistema. Il msgProcessor riceve il messaggio del Core e mette in esecuzione il programma di streaming e segnala l'avvenuta ricezione del messaggio.

Il processo con cui l'utente richiede lo stop dello streaming è simmetrico. Oltre ai comandi /stream e /stop, l'utente ha la possibilità di attivare e disat-

tivare il sistema tramite i comandi /activate e /deactivate. Questi tuttavia non bloccano il sistema ma si limitano ad ignorare i messaggi. Una soluzione ottimale è rappresentata dall'utilizzo degli Event messi a disposizione del modulo threading di python, ma non è ancora stata implementata.

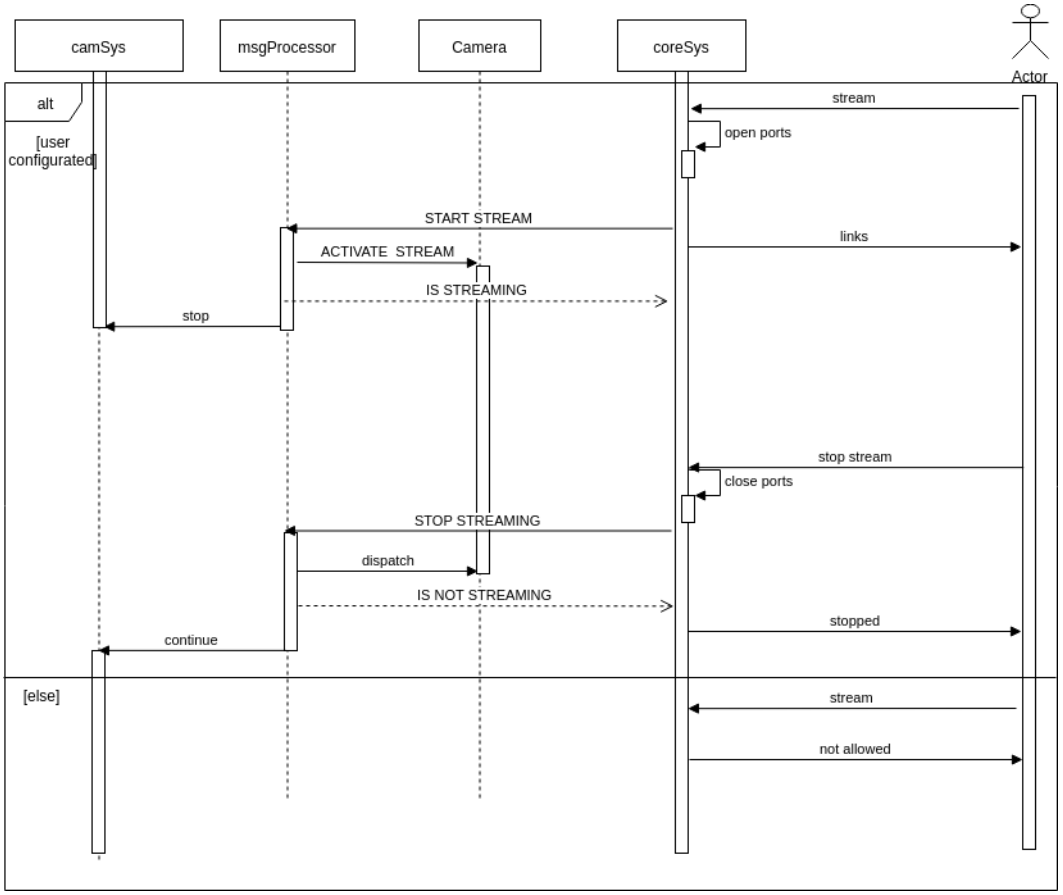


Figura 2.13: Diagramma delle sequenze relativo all'invio del comando di stream da parte dell'utente

Nel secondo caso, quello in cui un sensore PIR rileva del movimento, la situazione è illustrata in Figura 2.14. Viene inviato un messaggio di alert al coreSystem, questo risponde richiedendo l'attivazione dello streaming. Il msgProcessor gestisce il messaggio attivando lo streaming. Infine segnala al coreSystem che il comando è stato eseguito. Il coreSystem, dunque, richiede l'analisi dello streaming video. Se viene rilevata una sagoma umana, si

avverte l'utente con un messaggio contenente il nome della camera che ha generato l'alert. Quando l'utente richiede lo streaming, il servizio di analisi del video viene disabilitato.

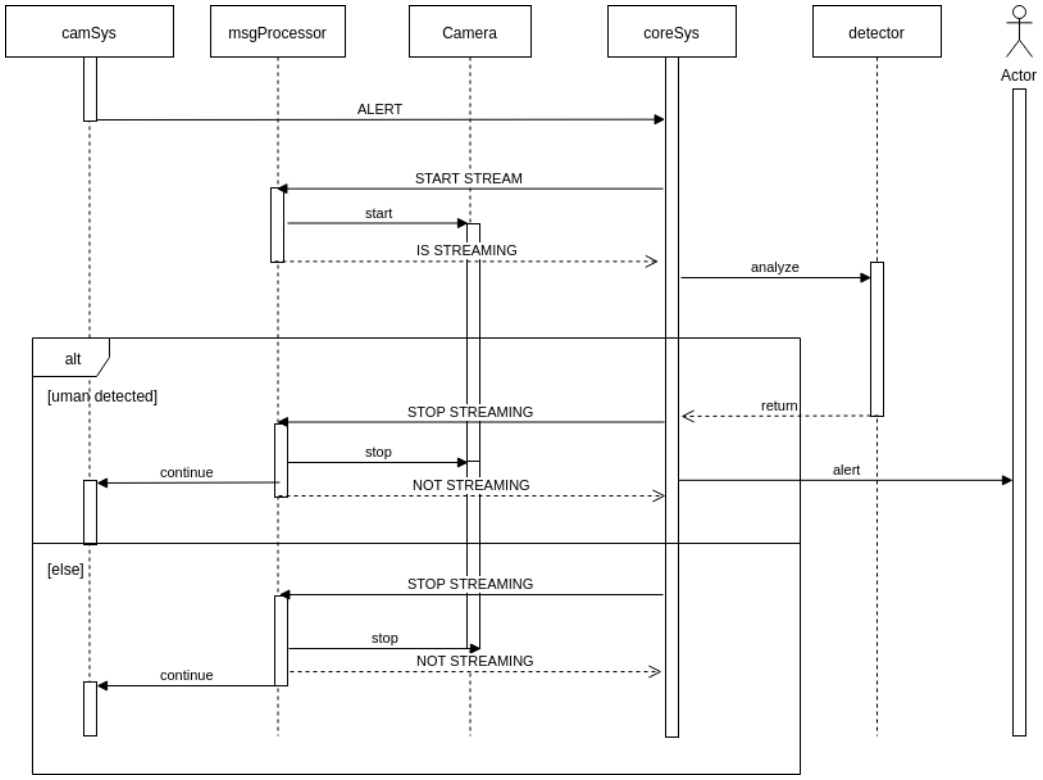


Figura 2.14: Diagramma delle sequenze relativo al sensore PIR che rileva del movimento

Si passa ora alla descrizione di alcune scelte implementative. I dispositivi Cam si connettono al broker come subscriber sul topic commands/[nome_camera]. Queste pubblicano messaggi sui topic executedcmd/[nome_camera] e detection/[nome_camera]. Il topic executedcmd è stato utilizzato per poter garantire che i comandi inviati dal Core vengano effettivamente eseguiti prima che questo proceda a fare altre operazioni.

La connessione con il server MQTT viene stabilita specificando il flag CleanSession a True. Questo impedisce al broker di salvare le informazioni relative alla sessione con il client come i topic a cui è sottoscritto, il tipo

di connessione instaurata e i messaggi non ancora inoltrati. La scelta è stata fatta per evitare che i client ricevano i messaggi inviati al broker mentre non erano connessi. Di fatto, se il dispositivo non è connesso, nell'ipotesi in cui dovesse rilevare un'intrusione, non sarebbe più utile. Lo svantaggio di questa soluzione è rappresentato dalla necessità di dover ristabilire la connessione TSL ad ogni disconnessione (Sezione 3.4). Tuttavia, essendo la disconnessione un caso poco frequente, si è ritenuto il costo accettabile.

L'operazione di subscribing viene effettuata richiedendo un un livello di QoS pari a 2. Nonostante tale scelta comporti un hoverhead maggiore, si vuole evitare il caso in cui il Core non riceva l'avviso o ne riceva più di uno relativo alla stessa rilevazione. Nei test effettuati non sono state notate differenze prestazionali rispetto all'uso di un Qos level 0.

Il Core invece, si registra come subscriber per i topic executedcmd/+ e detection/+. A differenza delle Cam, si è fatto uso di una wildcard per poter ricevere messaggi da tutti i client che pubblicano sui root topic executedcmd/ e detection/.

All'avvio del coreSystem, dopo aver inizializzato il client MQTT, viene inizializzato il Bot telegram e viene caricato in memoria il modello per fare il riconoscimento delle immagini. La fase di riconoscimento è quella più impegnativa per le risorse del Raspberry Pi, pertanto è importante averlo disponibile in memoria dall'avvio, per evitare ulteriori rallentamenti durante l'esecuzione dell'algoritmo.

Per poter leggere lo streaming video si utilizza la libreria OpenCV. Questa permette di estrarre i frame e convertirli per poter essere gestite dalla MobileNet, i parametri di normalizzazione sono stati specificati dagli autori dell'implementazione.

Capitolo 3

Messa in opera

3.1 Assemblaggio prototipo

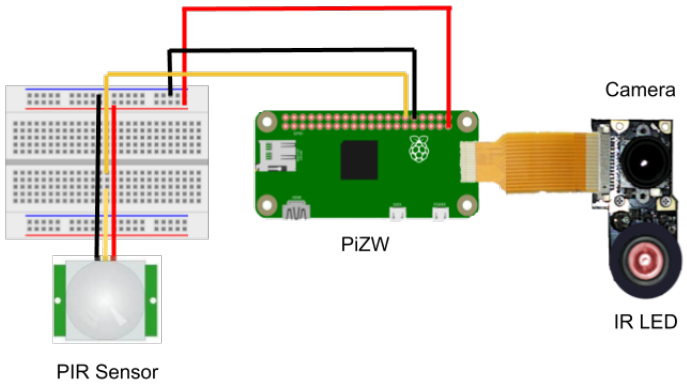


Figura 3.1: Prototipo IPCamera

Come già illustrato nella Sezione 2.2, il sistema è composto da due tipologie di dispositivo: la Cam e il Core.

Per quanto riguarda la Cam, un esempio di configurazione è quello esposto in Figura 3.1. Abbiamo un Raspberry Pi Zero W al quale sono connessi il modulo camera e il sensore PIR, entrambi alimentati direttamente dal Raspberry Pi.

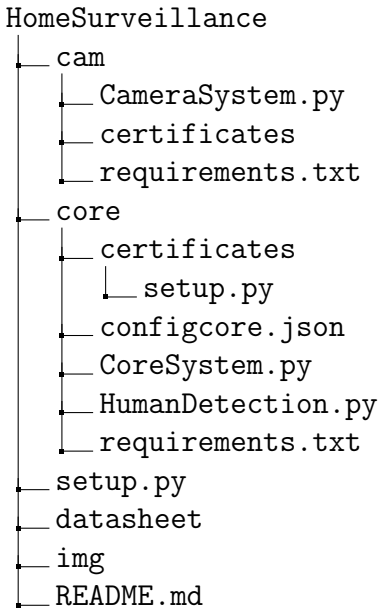


Figura 3.2: Vista directory del progetto

Nell'esempio il sensore PIR è connesso al secondo pin che offre una tensione di 5v, al nono pin per il ground e infine all'undicesimo pin per l'output.

Il modulo camera è connesso all'interfaccia CSI del Raspberry. Opzionalmente è possibile aggiungere due LED IR con fotocellula e nel setup di sviluppo si è scelto di montarne uno (la motivazione della scelta verrà discussa nella Sezione 3.3.2).

Per quanto riguarda il Core, non sono necessarie operazioni particolari.

3.2 Setup del sistema

Il sorgente del progetto, è disponibile all'indirizzo `https://github.com/AdamF42/HomeSurveillance.git`. La Figura 3.2 mostra la struttura del repository HomeSurveillance del progetto. A causa della mancanza di un installer, prima di poter sfruttare il codice, è necessario compiere qualche passaggio.

IP Statico

In prima istanza è necessario impostare un indirizzo IP statico su tutti gli apparati che si vuole connettere alla rete LAN. Generalmente, in una rete domestica basata sul protocollo IP, quando un dispositivo (client) intende connettersi richiede al router (server) un indirizzo che lo identifica univocamente all'interno della rete. Il protocollo che gestisce tale procedura è il Dynamic Host Configuration Protocol (DHCP). In sostanza il client DHCP richiede un indirizzo e il server DHCP si occupa di assegnarne uno ad ogni dispositivo fino all'esaurimento degli indirizzi disponibili. L'operazione viene svolta in maniera dinamica, ciò implica che ad ogni riconnessione, i client avranno un indirizzo diverso.

Dato che il sistema progettato fa uso del protocollo Transport Layer Security (TSL), devono essere generati dei certificati legati all'indirizzo IP di ogni dispositivo per permetterne l'autenticazione, pertatanto è necessario che ogni client richieda al server DHCP di assegnare sempre lo stesso indirizzo ad ogni riconnessione.

Nei sistemi Unix il gestore del protocollo client DHCP è il demone dhcpcd. Si dovrà quindi modificare la configurazione di questo accedendo al file /etc/dhcpcd.conf in ogni device che si intende utilizzare. Un esempio è mostrato in Figura 3.3, dove il campo **static ip_address** è l'indirizzo scelto per il dispositivo in esempio (ogni dispositivo avrà un valore diverso) e il campo **static routers** è l'indirizzo del router.

Creazione Telegram Bot

Per poter utilizzare il codice relativo al Telegram Bot è necessario crearne uno. Occorre aprire un client Telegram e ricercare l'utente BotFather. Una volta aggiunto ai contatti, verranno inviate sulla chat le informazioni utili per ricevere il codice identificativo (TOKEN).

```
interface eth0
static ip_address=192.168.1.222
static routers=192.168.1.250
static domain_name_servers=8.8.8.8

interface wlan0
static ip_address=192.168.1.222
static routers=192.168.1.250
static domain_name_servers=8.8.8.8
```

Figura 3.3: Esempio di configurazione del file dhcpcd.conf

File di configurazione

Il secondo passo da compiere è la compilazione del file HomeSurveillance/core/configcore.json. Quì andranno specificati gli indirizzi dei dispositivi connessi e le porte che si desidera utilizzare.

Installazione dipendenze

L'intero progetto è sviluppato in Python3, in particolare la versione 3.5.3. Quindi occorre installarlo sia sul Core che sulle Cam. All'interno sia della directory core, che di quella cam, vi è un file requirements.txt con la lista delle dipendenze installabili con l'utility pip. Un discorso a parte deve essere dedicato ad OpenCV (solo sul Core), Mosquitto (solo su Core), MJPG-Streamer (solo sulle Cam) ed Stunnel (su entrambi).

Mosquitto ed Stunnel sono presenti nei repository Debian e facilmente installabili con il comando apt-get. Per OpenCV[41] ed MJPG-Streamer[34] è necessario scaricare il sorgente e compilarlo.

Infine è necessario scaricare modello e prototipo da utilizzare con la MobileNets[36].

Creazione dei certificati

L'uso di certificati permette la cifratura dei messaggi sia per quanto riguarda il protocollo MQTT che quello HTTP.

Per tale operazione occorre affidarsi ad una Certification Autority (CA). Per via del modo in cui il sistema è pesato, la scelta più semplice è quella di generare una CA ed usarla per firmare tutti i certificati necessari (nella Sezione 3.4 verrà illustrato il meccanismo di funzionamento più nel dettaglio). Occorrerà un certificato per il Core e uno per ogni altra Cam che si vuole utilizzare. A tal scopo ci si è avvalsi di un'implementazione open-source dei protocolli SSL e TLS: OpenSSL[42]. Dalla directory HomeSurvelliance/core/certificates sul Core si eseguono i seguenti comandi:

```
openssl genrsa -out ca.key 4096
openssl req -new -x509 -days 3650 -key ca.key -out ca.crt
```

Con il primo comando viene generata la chiave privata mentre con il secondo viene utilizzata la stessa chiave per firmare il certificato radice. A questo punto, è possibile avvalersi dello script gencert.py. Questo in prima istanza legge il file di configurazione e genera un certificato per dispositivo in base all'indirizzo IP specificato. Inoltre genera i file di configurazione per stunnel: per il Core, tutti i dispositivi Cam saranno raggruppati in un unico file, come mostrato in Figura 3.2

Una volta terminato il comando, all'interno della directory HomeSurveillance/certificates comparirà una cartella per ogni Cam con all'interno certificato, chiave e file di configurazione per stunnel(Figura 3.5). Occorre copiare il contenuto della cartella all'interno della cartella HomeSurveillance/cam/-certificates presente nella Cam.

Avendo generato i certificati, è ora possibile configurare il server mosquitto modificando il file locato in /etc/mosquitto/mosquitto.conf (Figura 3.5). Si richiede al server mosquitto di accettare richieste sulla sola porta 8883, la porta di default per usare MQTT con TSL, inoltre viene richiesto di rifiutare ogni client non provvisto di certificato.

```
; ************************************************
; * Global options *
;************************************************
; Debugging stuff (may useful for troubleshooting)
debug = 7
output = stunnel.log
; ************************************************
; * Single cam options *
;************************************************

[cam01]
key = /home/pi/core/certificates/cam01/cam01.key
cert = /home/pi/core/certificates/cam01/cam01.cert
client = yes
accept = 8081
connect = 192.168.1.223:1234

[cam02]
key = /home/pi/certificates/cam02.key
cert = /home/pi/certificates/cam02.cert
client = yes
accept = 8083
connect = 192.168.1.223:1236
```

Figura 3.4: Esempio di configurazione di stunnel

```
cam01
├── cam03.cert
├── cam03.config
├── cam03.request
├── cam03.key
└── stunnel.conf
```

Figura 3.5: Esempio di HomeSurveillance/certificates/cam01.

```
# MQTT over TLS
listener 8883

cafile /path/to/generated/certificate/ca.crt

certfile /path/to/generated/certificate/core.crt

keyfile /path/to/generated/certificate/core.key

require_certificate true
```

Figura 3.6: Esempio di configurazione mosquitto

```
iptables -A INPUT -p tcp -s localhost --dport <porta di
   interesse> -j ACCEPT
iptables -A INPUT -p tcp --dport <porta di interesse> -j DROP
```

Figura 3.7: Comandi iptables

L'ultimo passaggio è quello di rendere la porta sulla quale scrive MJPG-streamer, accessibile solo dall'indirizzo 127.0.0.1 (localhost). Per farlo basta eseguire i comandi in Figura 3.5.

Avvio

Al momento non è ancora stato implementato uno servizio che gestisca l'avvio automatico del sistema al boot dei Raspberry Pi. Dunque, è necessario accedere al terminale di ogni dispositivo ed avviare Mosquitto, stunnel e infine gli script del sistema. Per Mosquitto, basta digitare il comando "service mosquitto start" sul Core. Per avviare stunnel bisogna fare in modo che questo utilizzi il file di configurazione creato dallo script. Basta posizionarsi nella cartella certificates di ogni dispositivo e dare il comando "stunnel stunnel.conf". A questo punto è possibile lanciare lo CoreSystem.py sul Core e

CameraSystem.py sulle Cam.

3.3 Test e Collaudo

3.3.1 Efficacia del sistema

L'efficacia del sistema è data da due fattori critici: la capacità di rilevare minacce e la velocità con cui viene notificato un utente.

Per stimare il primo dei fattori critici, è stato definito un test come segue:

- La Cam e stata posizionata lontano da fonti di disturbo come finestre, caloriferi e altri oggetti che possano ridurre la sensibilità del sensore PIR.
- I soggetti utilizzati per attivare il sensore sono rimasti nell'inquadratura della camera per almeno 3 secondi.
- Il numero massimo di frame che l'algoritmo può esaminare prima di ritornare, è stato settato a 15.

Il test è stato ripetuto 120 volte a distanza di 3, 5 e 7 metri sia in presenza che in assenza di luce. Inoltre la sensibilità del sensore è stata opportunamente regolata ad ogni cambiamento di distanza.

I risultati ottenuti sono esposti nella Tabella 3.1. Nonostante il numero di prove effettuate non sia sufficiente a trarre conclusioni definitive, si possono comunque muovere due osservazioni: il sistema non è influenzata dalle condizioni di luminosità; la capacità di rilevare minacce cala drasticamente a distanza di 7 metri. La prima osservazione è facilmente giustificabile osservando le riprese effettuate in condizioni notturne e diurne in Figura 3.8.

Per tentare di giustificare la seconda osservazione si è pensato di analizzare i file di log. Il numero di alert registrati nel file di log della Cam è congruo con il numero di segnalazioni ricevute dall'utente. Se ne può dedurre che la causa delle mancate segnalazioni è da attribuirsi al sensore PIR. Dunque il

Distanza		Numero segnalazioni all'utente/Numero rilevazioni
3 metri	Giorno	20/20
	Notte	20/20
5 metri	Giorno	19/20
	Notte	18/20
7 metri	Giorno	13/20
	Notte	12/20

Tabella 3.1: Tabella riassuntiva dei test

Figura 3.8: Riprese della Cam in condizioni di luce (sinistra) e senza (destra)

sistema lavora in modo ottimale solo se l'eventuale intrusione si verifica al di sotto dei 7 metri.

Tuttavia esistono due situazioni in cui il sistema potrebbe non rilevare l'intrusione: nel caso di un falso positivo seguito da un'intrusione mentre si utilizzano più camere; nel caso in cui l'algoritmo fallisce il riconoscimento al primo frame utile.

La prima si verifica quando un falso positivo generato da una camera impegna il Core nell'analisi del video. Se, nello stesso momento, una seconda Cam rileva un'intrusione, il messaggio di alert verrà esaminato solo dopo la conclusione dell'analisi. Nel momento in cui il Core passa all'analisi dello stream della seconda Cam, il soggetto potrebbe non essere più presente nell'inquadratura. Al momento il problema è stato arginato riducendo a 15 il numero massimo di frame che possono essere processati. Una soluzione ottimale è utilizzare un thread separato nel Core per raccogliere i messaggi MQTT. Una volta ricevuto l'alert, si richiede alla Cam di salvare i frame. Non appena il thread che si occupa dell'analisi del video torna libero, si richiede alla Cam di inviare i frame salvati.

La seconda è strettamente legata al meccanismo di funzionamento dell'algoritmo di riconoscimento. L'algoritmo lavora in due fasi: lettura dei frame ed elaborazione di questi. La prima operazione è relativamente veloce, mentre la seconda blocca il thread per diverso tempo. Il tempo di elaborazione di un frame è di circa un secondo. Se il soggetto non viene riconosciuto alla prima lettura ed esce fuori dall'inquadratura della camera, il frame seguente non conterrà più la sagoma dell'intruso.

Una possibile soluzione è quella di caricare i frame letti dallo stream video in una coda e svuotarla in un sotto-processo separato che si occuperà di analizzarli. Tuttavia, la soluzione presentata, non è ancora stata ancora implementata.

Per quanto concerne le tempistiche di notifica dell'utente, si è deciso di analizzare i file di log. Come già illustrato, quando il sensore PIR di una Cam viene attivato, questa manda un messaggio di alert al Core. Sarà poi

compito del Core analizzare il video ed eventualmente avvisare l'utente.

I momenti che possono rallentare la notifica dell'utente sono quattro: il tempo che intercorre tra l'attivazione del PIR e l'invio dell'alert; il tempo tra l'invio e ricezione del'alert; il tempo tra la ricezione dell'alert e il riconoscimento da parte dell'algoritmo; il tempo tra l'avvenuto riconoscimento e l'invio del messaggio dal Telegram Bot all'utente.

Per il primo punto, tra l'attivazione del sensore PIR (Motion Detected!) e l'invio del messaggio di alert al Core (Alert Sent), intercorrono pochi decimi si secondo, come mostrato dalla Figura 3.9.

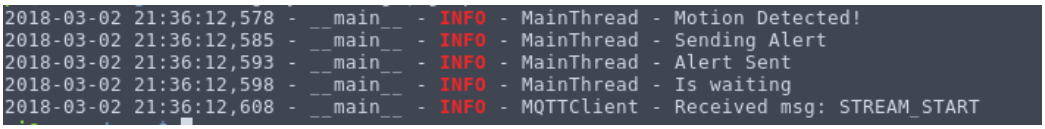


Figura 3.9: File di log della Cam

Per il secondo punto, sfortunatamente non si è a disposizione dei mezzi per poter effettuare una misurazione precisa. Tuttavia è possibile fornire una stima osservando il tempo trascorso per tra l'invio dell'alert da parte della Cam (Alert Sent) e la ricezione del comando di inizio streaming (Received msg: STREAM_START). Anche in questo caso, come mostrato in Figura 3.9, si è nell'ordine dei decimi di secondi.

Il terzo punto è fortemente dipendente dal tempo massimo che occorre al sistema per verificare un'intrusione. Questo è determinato dal numero massimo di frame che l'algoritmo processa. Per effettuare i test, questo valore è stato impostato a 15, il che comporta un tempo massimo di circa 13,5 secondi. Di fatto il numero di frame per secondo (FPS) che l'algoritmo di riconoscimento riesce ad analizzare è pari a circa 0.9 FPS. La misura è stata effettuata utilizzando la funzione FPS() messa a disposizione dalla libreria per python imutils[27]. Nel caso medio il tempo necessario al riconoscimento è di circa 3 secondi. Di fatto, nell'esempio in Figura 3.10, il tempo tra ricezione dell'alert (Received: "ALERT": "MOTION_DETECTED") e l'avvenuto riconscimento (Intrusion Detected) è di circa 2,5 secondi.

Infine, come testimoniato dall'esempio in Figura 3.10, i file di log confermano che tra l'avvenuto riconoscimento (Intrusion detected) e l'invio del messaggio da parte del Bot Telegram (Sent alert from camera), intercorrono pochi centesimi di secondo.

```
2018-03-03 04:07:19,117 - __main__ - INFO - TelegramBot - System Activated
2018-03-03 04:07:20,358 - __main__ - INFO - MQTTCoreCli - Received: {"ALERT": "MOTION_DETECTED"} from cam01
2018-03-03 04:07:20,584 - __main__ - INFO - MQTTCoreCli - The Cam cam01 has succesfully started stream
2018-03-03 04:07:20,586 - __main__ - INFO - MQTTCoreCli - Analizing video
2018-03-03 04:07:22,841 - __main__ - INFO - MQTTCoreCli - Intrusion detected
2018-03-03 04:07:23,060 - __main__ - INFO - MQTTCoreCli - Sent alert from camera cam01
```

Figura 3.10: File di log del Core

Il sistema dunque si è dimostrato essere sufficientemente efficace in condizioni di uso reali.

3.3.2 Analisi dei Consumi

Per poter misurare il consumo energetico è stato utilizzato un tester USB marchiato Muker[56]. Invece per analizzare l'assorbimento della CPU, è stata utilizzata l'utility htop. Le analisi sono state circoscritte ai soli programmi utilizzati dal sistema. Inoltre, è stato disattivato il service del bluetooth per entrambi i dispositivi ed è stata disabilitata la scheda WiFi per il Core (connesso con cavo Ethernet).

Per semplicità è stata suddivisa l'analisi in base allo stato in cui si possono trovare i dispositivi: abilitato e disabilitato.

Con il sistema disabilitato, la Cam utilizza poco più dell'uno percento della CPU, dovuto all'esecuzione del cameraSystem (Figura 3.11), con un assorbimento energetico di 0.55 Watt. Per quanto riguarda il Core, l'uso della CPU è praticamente nullo (Figura 3.12) e il consumo energetico si attesta sugli 1.22 Watt.

Bisogna distinguere due casi quando il sistema viene abilitato: l'utente che sta accedendo allo streaming e il Core che sta processando il video.

Il Core nel primo caso utilizza circa il 10% della CPU. In Figura 3.13, si può notare come il principale responsabile sia stunnel. Utilizzando due camere e visualizzando dei due dispositivi contemporaneamente, l'impatto di

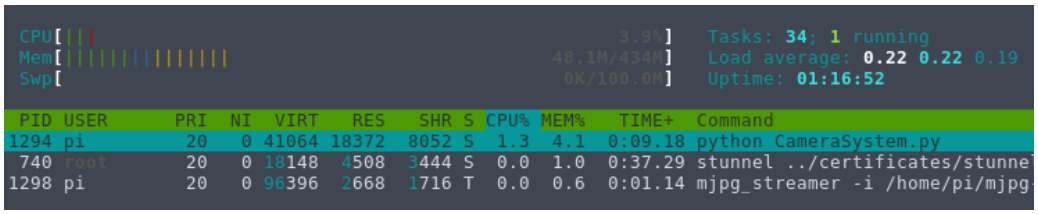


Figura 3.11: Cam in idle.

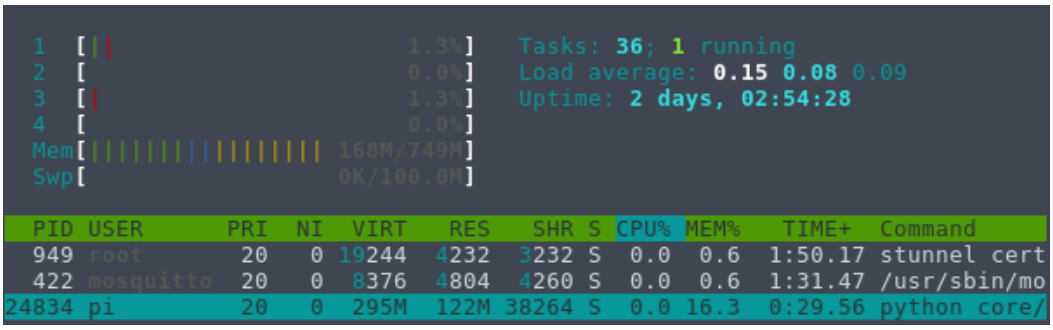


Figura 3.12: Core in idle.

stunnel raddoppia (Figura 3.14). Anche l'assorbimento energetico passa da 1.34 W (per una sola camera) a 1.69 Watt (nel caso di due camere).

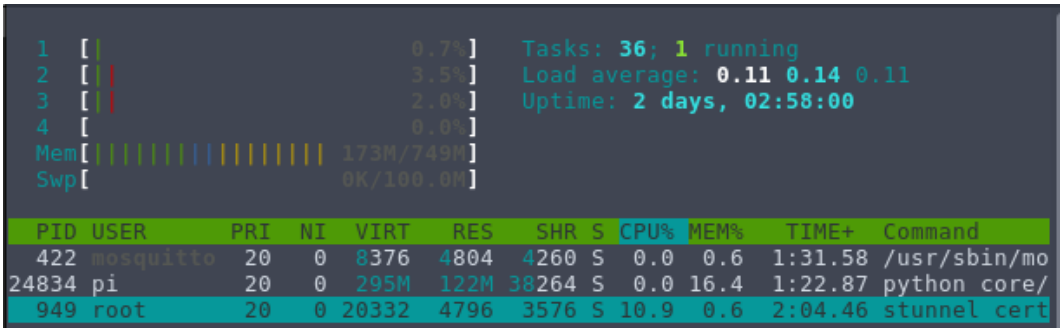


Figura 3.13: Core in streaming con una camera.

Nel secondo caso, quando viene eseguito l'algoritmo di riconoscimento, l'utilizzo della CPU raggiunge l'ottanta percento (Figura 3.15). In questa situazione, si registra il picco di assorbimento di 4 Watt.

Infine, sia nel primo che nel secondo caso la Cam ha registrato il picco massimo utilizzo del ventidue percento con un assorbimento di 1.2 Watt. Co-

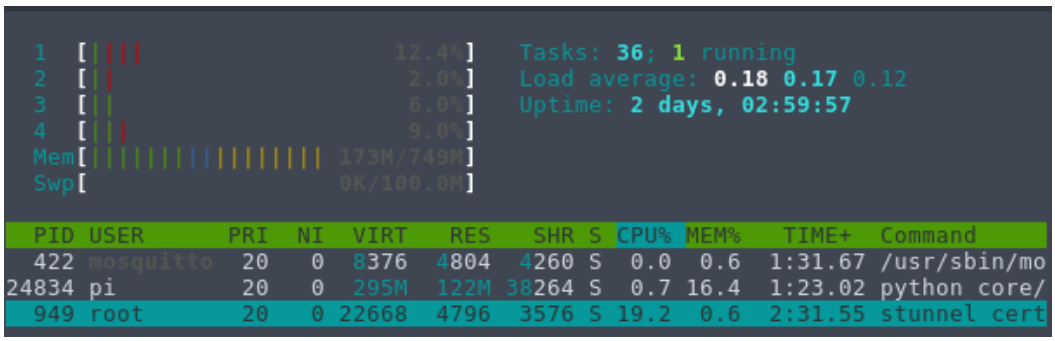


Figura 3.14: Core in streaming con due camere.

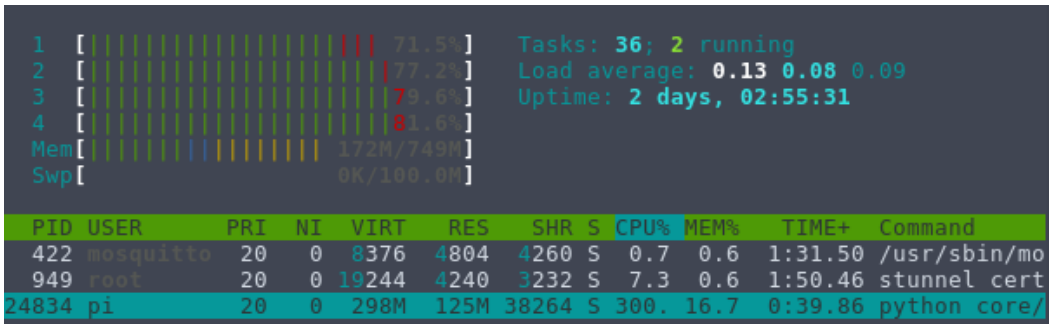


Figura 3.15: Core durante l'esecuzione dell'algoritmo di riconoscimento.

me visibile in Figura 3.16, anche in questo caso, la grossa parte del consumo è dovuta all'utilizzo di HTTPS mediante stunnel.

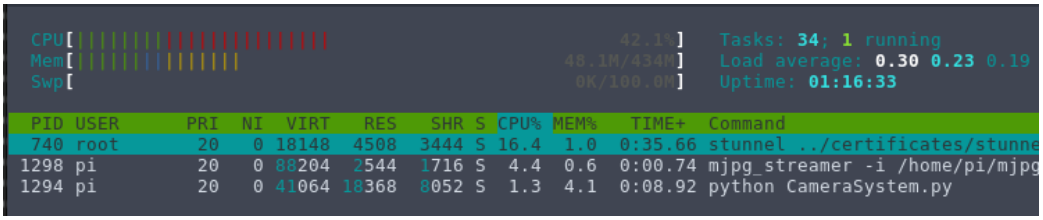


Figura 3.16: Camera in streaming.

Inoltre, ai dispositivi Cam bisogna aggiungere altri 0.58 Watt assorbiti dal led IR.

Per concludere, il consumo energetico è fortemente dipendente dal numero di rilevazioni che vengono effettuate ed è pesantemente influenzato dall'uso di crittografia su HTTP. Si potrebbe pensare di rimuovere lo strato di crittografia per abbattere i consumi, ma nella sezione seguente verrà illustrato il motivo per cui non è stata fatta questa scelta.

3.4 Considerazioni sulla sicurezza

Essendo il sistema esposto pensato per essere costantemente connesso alla rete Internet, l'aspetto di sicurezza che è importante valutare, riguarda le comunicazioni tra i dispositivi e l'utente. In questo ambito, per sicurezza si intende l'insieme delle tecnologie volte a garantire che un messaggio ricevuto dalla rete[12]:

- non è stato alterato (integrità)
- non è stato letto da chiunque (privacy)
- proviene da un mittente certo (autenticazione)
- contiene una prova che sia stato in mittente ad iniziare la comunicazione.

Come già illustrato, per garantire la sicurezza delle comunicazioni tra Cam, Core e broker si è scelto di utilizzare il protocollo Transport Layer Security (TSL). TSL è interposto fra lo strato applicativo e quello di trasporto: accetta richieste dai client e le manda al TCP perché siano trasmesse al server. Nel caso delle comunicazioni su MQTT, tutti i dispositivi del sistema sono client e il broker fa da server, mentre nel caso dello streaming video, le Cam fungono da server e il Core funge da client. Il TSL è composto da due sub-protocolli: uno per instaurare connessioni sicure e uno per utilizzarle[52]. La prima fase avviene durante l'handshake tra client e server. In sostanza, client e server si scambiano i certificati e verificano se la firma appartiene ad una Certification Authority (CA) nota. La seconda fase è quella in cui viene generata una chiave simmetrica per poter continuare ad utilizzare il canale creato senza l'onere di dover ripetere la procedura di autenticazione e validazione ad ogni messaggio.

Si sarebbero potuti utilizzare i soli certificati dei server e una coppia <nome_utente,password> per l'autenticazione dei client a livello applicativo. Di fatto sia MQTT che stunnel prevedono un meccanismo di autenticazione basato su nome utente e password. Tuttavia si è optato per utilizzare il meccanismo di autenticazione reciproca in quanto permette al server di chiudere il collegamento durante la procedura di handshake, quindi prima di aver stabilito la connessione. Quando vengono scambiati messaggi MQTT su TSL, è possibile specificare il flag ClientSession per permettere al broker di salvare la session key. Ma come spiegato nella Sezione 2.4, potrebbe dare origine a problemi. Per quanto riguarda invece le comunicazioni HTTPS, stunnel permette di salvare la session key solo se viene avviato in modalità demone[51].

Oltre al costo computazionale, un altro grande svantaggio nell'utilizzo di TSL è dovuto al sistema con cui i certificati possono essere distribuiti e revocati. Al momento tale operazione viene fatta copiando i certificati generati in ogni dispositivo come spiegato nella Sezione 3.2. Inoltre non è stata implementata un sistema di revoca.

Il motivo che rende l'uso del protocollo TSL così importante è che usando il solo protocollo TCP/IP i pacchetti dati scambiati tra due nodi, sono visibili da ogni soggetto in grado di introdursi nella rete e leggere o modificare i dati scambiati[44].

Allo stato attuale è stata gestita la sicurezza per le sola rete locale. Si potrebbe pensare che aggiungere uno strato di sicurezza ad una rete già protetta sia superfluo, tuttavia recenti ricerche hanno dimostrato la vulnerabilità del protocollo WPA2 [29]. Nonostante molte aziende abbiano già risolto il bug, l'aggiornamento non è stato rilasciato per la maggior parte dei dispositivi più datati. Inoltre spesso l'utente non è in grado di installare l'aggiornamento o non è a conoscenza della problematica. Si pensi ad esempio al caso in cui un attaccante si posizioni nel raggio d'azione del segnale WiFi del router, sarebbe possibile intercettare i frame video o manipolare i messaggi MQTT prima che questi giungano al Core rendendo il sistema inoffensivo. Inoltre la sola intercettazione dei frames potrebbe fornire informazioni utili ad eventuali malintenzionati.

Sfortunatamente non è stata gestita la sicurezza per quanto riguarda le connessioni al di fuori della rete privata. Di fatto il Core ridireziona lo streaming di ogni Cam su l'indirizzo esterno del router in chiaro. Una soluzione pensata è quella di generare un certificato associato all'indirizzo WAN ogni volta che viene persa la connessione con il router (se la connessione è stata persa, allora è probabile che il router si sia riavviato e quindi abbia un indirizzo nuovo) e poi gestire l'autenticazione a livello applicativo. Bisogna tuttavia considerare l'impatto che potrebbe avere sulle prestazioni del sistema. Ad ogni modo, questo aspetto non è stato considerato decisivo in quanto l'accesso alla rete locale non è permanente.

Un ultimo aspetto da considerare è l'utilizzo del Bot Telegram. Questo scambia messaggi con i server dell'azienda usando HTTPS, ma alla versione attuale, non è possibile utilizzare la crittografia end-to-end che è stata invece implementata per per le comunicazioni tra gli utenti normali[53]. In sostanza bisogna avere fiducia sul grado di riservatezza di Telegram.

Conclusioni e Sviluppi futuri

Con questo progetto, si è voluto studiare una possibile soluzione per un sistema di sorveglianza domestico a basso consumo e utilizzabile in ambienti con presenza di animali domestici.

Si è partiti dall'analisi dei sistemi esistenti per poi elaborare una soluzione che fosse il più possibile open-hardware e open-software. Di fatto vengono utilizzate delle schede Raspberry Pi (open-hardware) e un insieme di librerie e programmi open-source.

Inizialmente è stato illustrato il funzionamento del sistema, come assemblare un prototipo e i passi necessari per ed effettuare il primo avvio. Successivamente si è studiata l'efficacia e l'efficienza del sistema. Nonostante risultati ottenuti siano stati soddisfacenti nella maggior parte delle situazioni, oltre le problematiche illustrate, il sistema presenta ampi margini di miglioramento. Si potrebbe aggiungere un'interfaccia web per migliorare l'interazione con l'utente, studiare una gestione dei thread più intelligente per eliminare cicli non necessari, utilizzare un solo port forwarding per tutte le camere e aggiungere una funzione per il salvataggio delle riprese in locale. Inoltre, grazie all'utilizzo del protocollo MQTT, con poche modifiche si potrebbe utilizzare dei servomotori per direzionare le camere da remoto.

Bibliografia

[1] John R. Ackermann. Toward open source hardware. http://www.tapr.org/Ackermann_Open_Source_Hardware_Article_2009.pdf.

[2] K.A.M Annuar, N.A. Ab Hadi, S.K. Subramaniam, M. F. Mohd Ab Halim, A.F. Kadmin W.N. Abd Rashid, M.S. Amri, and A. Abdul Salam. Intelligent image capturing alarm system using raspberry pi. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 2017.

[3] Steve Ballmer. Microsoft ceo takes launch break with the sun-times. https://web.archive.org/web/20011108013601/http://www.suntimes.com/output/tech/cst-fin-micro01.html.

[4] Giuseppe Costa. I retailer possono risparmiare molto con la business intelligence open source. https://blog.extrasys.it/it/smartblog/i-retailer-possono-risparmiare-molto-con-la/-business-intelligence-open-source.

[5] Security News Desk. Ip security camera and network video surveillance visionary - security news desk. http://www.securitynewsdesk.com/ip-security-camera-and-network-video-surveillance-visionary/.

[6] Eclipse. Mosquitto. https://mosquitto.org/.

[7] Eclipse. Paho - mqtt and mqtt-sn software. https://www.eclipse.org/paho/.

[8] Eclipse. paho-mqtt 1.3.1 : Python package index. https://pypi.python.org/pypi/paho-mqtt#network-loop.

[9] Andrea Bonaccorsi et al. Why open source software can succeed. *Research Policy*, 2003.

[10] Andrew G. Howard et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 2017.

[11] Ben Nuttall et al. 12. api - input devices — gpiozero 1.4.1 documentation. http://gpiozero.readthedocs.io/en/stable/api_input.html.

[12] Fred Halsall et al. *Computer Networking and the Internet: 5th (fifth) Edition*, chapter 6. Addison Wesley, 3006.

[13] Joseph Redmon et al. You only look once: Unified, real-time object detection. 2016.

[14] JWei Liu et al. Ssd: Single shot multibox detector. 2016.

[15] Sandeep Krishnamurthy et al. An analysis of open source business models. *University of Washington Bothell*, February 2003.

[16] Shaoqing Ren et al. Faster r-cnn: Towards real-time object detection with region proposal networks. 2016.

[17] Ziyuan Wang et al. Security and privacy issues within the cloud computing. *International Conference on Computational and Information Sciences*, 2011.

[18] EZVIZ. Video surveillance, security systems, sports cameras. https://ezviz.eu/.

[19] Foscam c2. https://www.foscam.it/foscam_it/foscam-c2-2-megapixel-full-hd1080p-h-264-wireless-cavo-con\-filtro-ir-cut-8-metri-120.html.

[20] Free Software Foundation. Cos'è il software libero? https://www.gnu.org/philosophy/free-sw.it.html.

[21] Raspberry Pi Foundation. Raspberry pi 3 model b - raspberry pi. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[22] Samuel Greengard. *The Internet of things*. Massachusetts Institute of Technology, 2015.

[23] Datasheet hc-sr501. https://www.mpja.com/download/31227sc.pdf.

[24] Frank Hecker. Setting up shop: The business of open-source softwar.

[25] Harry Henderson. *Encyclopedia of Computer Science and Technology*.

[26] Gaston C. Hillar. *MQTT Essentials - A Lightweight IoT Protocol*. 2017.

[27] imutils/fps_demo.py at master · jrosebr1/imutils. https://github.com/jrosebr1/imutils/blob/master/demos/fps_demo.py.

[28] Brandon Joffe. Home surveillance system with facial recognition. https://github.com/BrandonJoffe/home_surveillance.

[29] Krack attacks: Breaking wpa2. https://www.krackattacks.com/.

[30] Kumantech. 5mp camera module. http://www.kumantech.com/kuman-5mp-1080p-hd-camera-module-for-raspberry-pi-for-rasp\berry-pi-3-model-b-b-a-rpi-2-1-sc15_p0063.html.

[31] Keith W. Ross James F. Kurose. *Internet e reti di calcolatori*, chapter 1. 2003.

[32] Bernard Marr. A short history of machine learning - data science central. https://www.datasciencecentral.com/profiles/blogs/a-short-history-of-machine-learning.

[33] Virginia Menezes, Vamsikrishna Patchava, and M. Surya Deekshith Gupta. Surveillance and monitoring system usingraspberry pi and simplecv. pages 1276–1278. International Conference on Green Computing and Internet of Things (ICGCIoT), IEEE, 2015.

[34] MGPG-Streamer. jacksonliam/mjpg-streamer. https://github.com/jacksonliam/mjpg-streamer.

[35] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.

[36] chuanqi305/mobilenet-ssd: Caffe implementation of google mobilenet ssd detection network, with pretrained weights on voc0712 and map=0.727. https://github.com/chuanqi305/MobileNet-SSD.

[37] https://it.wikipedia.org/wiki/MQTT.

[38] Mqtt version 3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[39] NETGEAR. Arlo datasheet. https://www.arlo.com/it/images/documents/arlo_datasheet.pdf.

[40] Huu-Quoc Nguyen, Ton Thi Kim Loan, Bui Dinh Mao, and Eui-Nam Huh. Low cost real-time system monitoring using raspberry pi. *Ubiquitous and Future Networks (ICUFN), 2015 Seventh International Conference on*, 2015.

[41] Opencv library. https://opencv.org/.

[42] Openssl website. https://www.openssl.org/.

[43] USB Organization. Usb.org frequently asked questions. http://www.usb.org/developers/usbfaq#cab2.

[44] Tanmay Patange. How to defend yourself against mitm or man-in-the-middle attack. https://hackerspace.kinja.com/how-to-defend-yourself-against-mitm-or-man-in-the-middl.

[45] Piroelettricità - wikipedia. https://it.wikipedia.org/wiki/Piroelettricit

[46] Port forwarding definition from pc magazine encyclopedia. https://www.pcmag.com/encyclopedia/term/49509/port-forwarding.

[47] Tomi D. Raty. Survey on contemporary remote surveillance systems for public safety. *IEEE Transactions on Systems, Man,and Cybernetics*, 2010.

[48] Tom Shanley. *Plug and play system architecture*. Reading, Mass. : Addison-Wesley, 1995.

[49] Reuters Staff. Russia's zuckerberg launches telegram, a new instant messenger service. https://www.reuters.com/article/idUS74722569420130830.

[50] stunnel: Home. https://www.stunnel.org/.

[51] stunnel: Howto. https://www.stunnel.org/howto.html.

[52] A. S. Tanenbaum. *Reti di Calcolatori*. 2011.

[53] Telegram. Telegram bot api. https://core.telegram.org/bots/api.

[54] Telegram. Telegram f.a.q. https://telegram.org/faq#q-what-is-telegram-what-do-i-do-here.

[55] python-telegram-bot. https://python-telegram-bot.org/.

[56] Usb safety tester j7-t. http://lygte-info.dk/review/USBmeter%20safety%20tester%20J7-t%20UK.html.

[57] M. Valera and S. Velastin. Intelligent distributed surveillance systems: a review. *IEE Proceedings - Vision, Image and Signal Processing*, 2005.

Ringraziamenti

Ringrazio il professore Renzo Davoli per il tempo dedicatomi.
Ringrazio la mia famiglia per avermi sostenuto e per avermi permesso di essere dove sono.
Infine ringrazio tutti coloro che mi hanno sopportato le giornate più stressanti.